Sistemas Digitais

Sequenciamento e Controle
ASM – *Algorithmic State Machine*

Referência Bibliográfica:

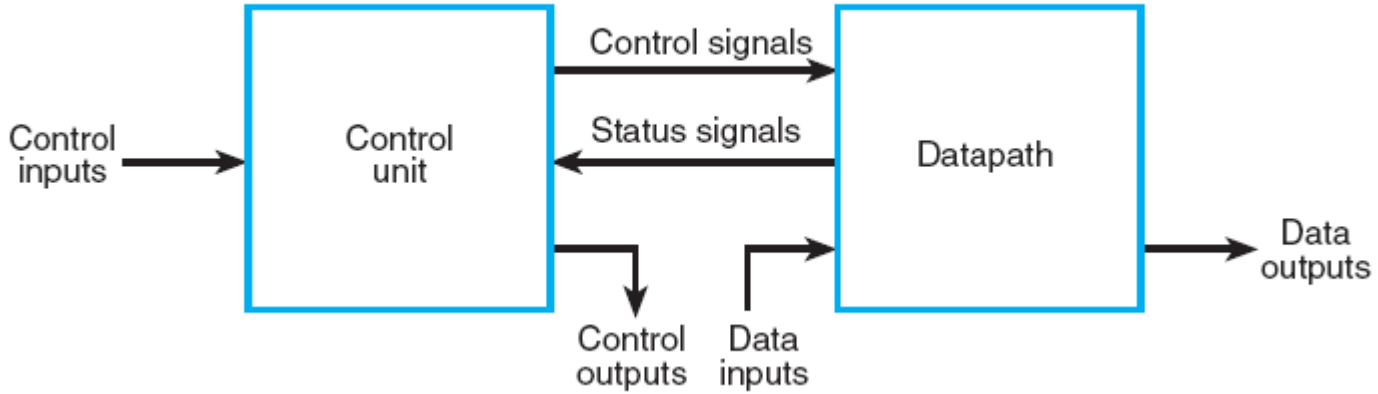
Logic and Computer Design Fundamentals – Mano & Kime

Adaptação: josé artur quilici-gonzalez

Sumário

- Interação entre Caminho de Dados e Unidade de Controle
- Máquina de Estado Algorítmica -*ASM*
  - Diagrama *ASM*
  - Considerações de *Timing*
- Exemplo de Diagrama *ASM*
  - Multiplicador Binário
- Controle *Hardwired*
  - Métodos de Projeto de Controle
    - Registrador de Sequência e Decodificador
    - Um *Flip-Flop* por Estado
- Controle Microprogramado

Interação entre *Datapath* e Unidade de Controle



- **Caminho de Dados *(Datapath)*** – realiza transferências e processamento de dados
- **Unidade de Controle** – determina a sequência de operações no *Datapath*

- **Sinais de Estado** descrevem propriedades do estado do *Datapath*
- A **Unidade de Controle** recebe **Sinais de Estado** do ***Datapath*** (úteis para determinar a sequência específica de operações)
- Tanto a **Unidade de Controle** quanto o **Caminho de Dados** (***Datapath***) podem interagir com outras partes do **Sistema Digital** (como Memória, Lógica de *Input-Output* etc.) através das **Entradas e Saídas de Dados e Controles** etc.

Tipos de Unidade de Controle

- Há duas classes distintas:
  - Unidade de Controle **Programável**
  - Unidade de Controle **Não-programável *(Hardwired Control)***
- Uma **Unidade de Controle Programável** possui:
  - Um **Contador de Programa** (PC) (*Program Counter*) ou outro **Registrador de Sequência**, cujo conteúdo aponta para a próxima instrução a ser executada
  - Uma ***ROM*** ou ***RAM*** externa para guardar instruções e informação de controle
  - Uma **Lógica de Decisão** (baseada em Sinais de Estado) para determinar a sequência de microoperações, e lógica para interpretar as instruções
- Uma **Unidade de Controle Não-programável** não busca instruções em uma memória e não possui um mecanismo para determinar a sequência de execução dessas instruções. Essa Unidade de Controle determina quais operações serão executadas e em que sequência, baseando-se unicamente nos *bits* de Entrada e de Estado
  - Este tipo de Unidade de Controle será visto mais detalhadamente agora, tendo como exemplo um **Multiplicador**

Representação para Unidade de Controle

- Uma representação possível para o projeto de Unidades de Controle é o **Diagrama *ASM*** *(Algorithmic State Machine)*

- Normalmente, a parte do Projeto Digital mais difícil e criativa é a formulação do **Algoritmo de *Hardware***, que realiza os objetivos do processamento de dados

- Um **Algoritmo de *Hardware*** pode ser usado como base para definir tanto o **Caminho de Dados** como a **Unidade de Controle** do Sistema

ASM - *Algorithmic State Machine*

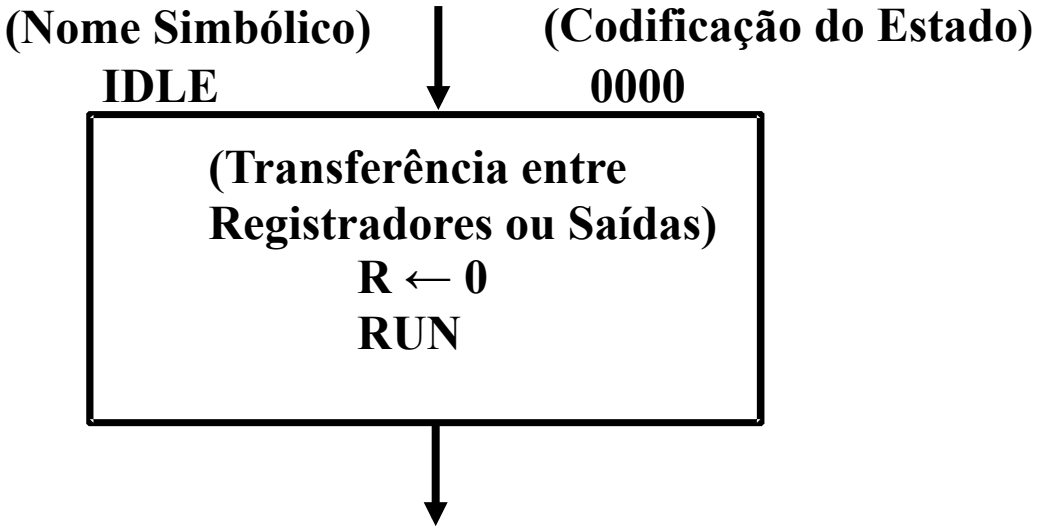
- A função de uma **Máquina de Estado** (ou de um circuito sequencial) pode ser representada por uma **Tabela de Estado** ou um **Diagrama de Estado**
- Um **Fluxograma** é uma forma de especificar o **fluxo de ações** e **controle** em um algoritmo
- Uma **Máquina de Estado Algorítmica** (***Algorithmic State Machine***) (***ASM***) é simplesmente um tipo conveniente de **fluxograma** usado para especificar **Diagramas de Estados** para **Lógica Sequencial** e, opcionalmente, **ações** realizadas no **Caminho de Dados** (***Datapath***)
- Ainda que um fluxograma, normalmente, não especifique o "tempo", uma *ASM* explicitamente especifica uma **Sequência de Ações** e suas **relações com o tempo**

Primitivas Usadas numa *ASM*

- A **Caixa de Estado** é um **retângulo**, marcada com o nome do estado, contendo transferências de registradores e sinais de saída ativados enquanto a Unidade de Controle estiver neste estado
- A **Caixa de Decisão Escalar** é um **losango** que descreve os efeitos de uma condição de entrada específica sobre o controle. Possui **uma entrada e duas saídas**, uma para **TRUE (1)** e outra para **FALSE (0)**
- A **Caixa de Decisão Vetorial** é um **hexágono** que descreve os efeitos de um vetor específico de *n-bits* ($n > 1$) das condições de entrada sobre o controle. Possui uma entrada e até 2^n saídas, cada uma correspondendo a um valor binário do vetor
- A **Caixa de Saída Condicional** tem a **forma ovalada**, com uma entrada vinda de uma Caixa de Decisão e saídas ativadas para as condições de decisão que devem ser satisfeitas

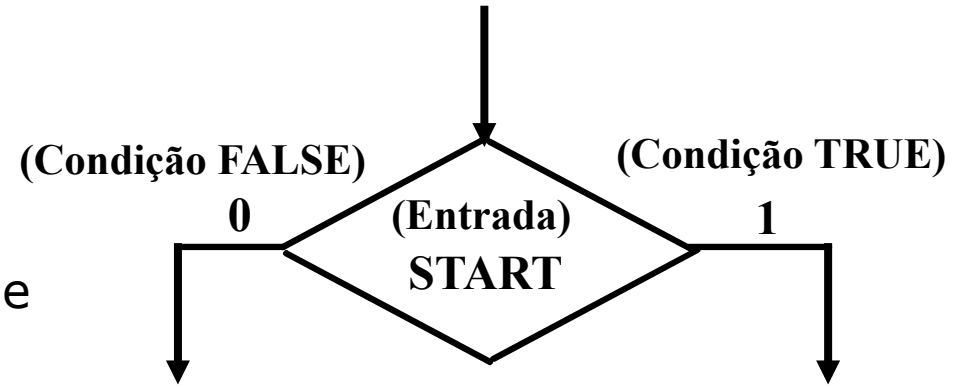
Caixa de Estado

- Um **retângulo**:
  - Com o **nome simbólico** do estado marcado no canto superior esquerdo
  - Contendo **operações de transferência** entre registradores e saídas ativadas enquanto a Unidade de Controle estiver neste estado
  - Uma **codificação de estado** opcional, se definida, na parte externa superior direita



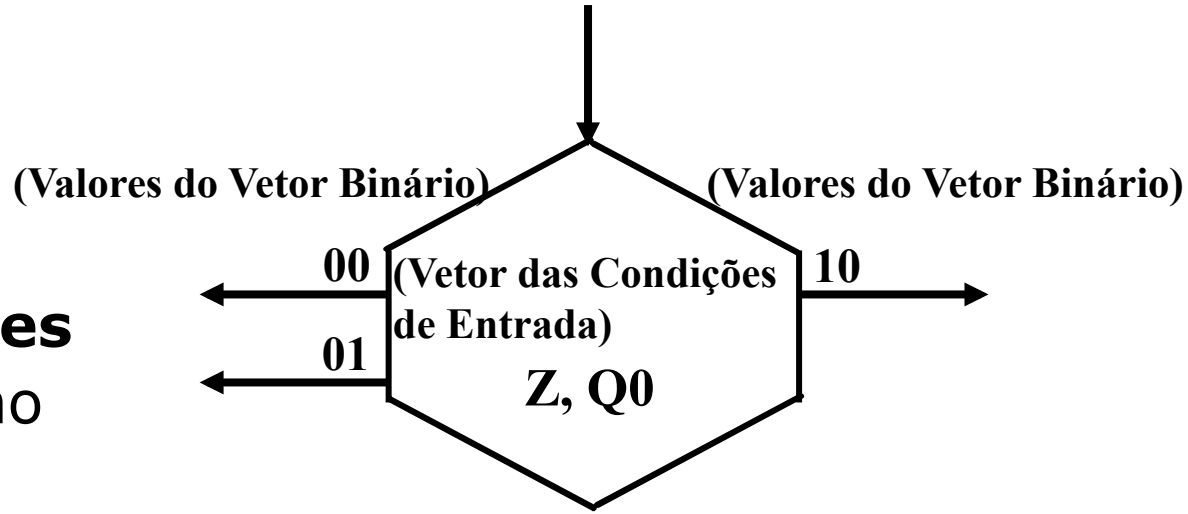
Caixa de Decisão Escalar

- Um **losango** com:
  - Uma **entrada**
  - Uma **condição de entrada**, colocada no centro da caixa, que é testada
  - Uma **saída TRUE** para quando a condição for verdadeira (valor lógico 1)
  - Uma **saída FALSE** para quando a condição for falsa (valor lógico 0)



Caixa de Decisão Vetorial

- Um **hexágono** com:
  - Uma **entrada**
  - Um **vetor das condições de entrada**, colocado no centro da caixa, que é testado
  - Até ***2ⁿ* saídas**. Os caminhos têm um valor de vetor binário que corresponde à condição satisfeita do vetor de entrada

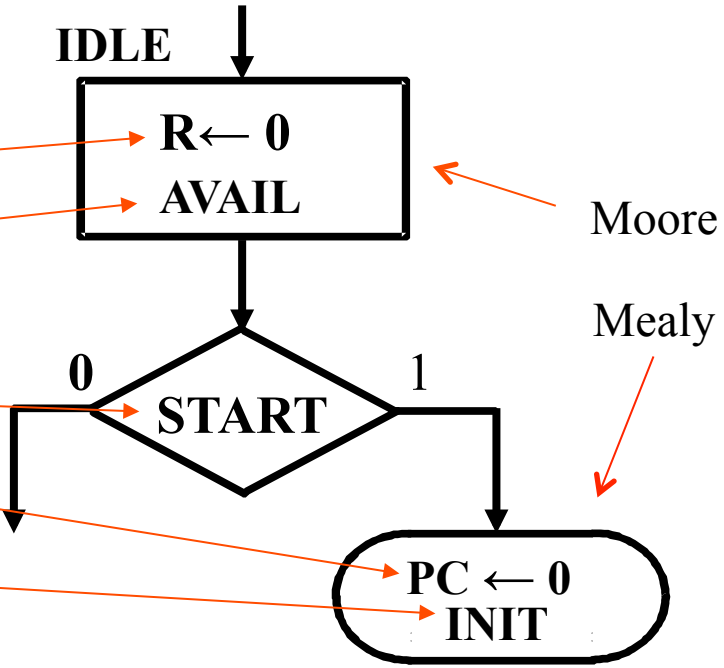


Caixa de Saída Condicional

- Um **retângulo ovalado** com:
  - Uma **entrada** vinda de uma Caixa ou Caixas de Decisão
  - Uma **saída**
  - **Transferências entre registradores** ou saídas que ocorrem somente se o caminho condicional da caixa foi o escolhido
- Transferências e saídas numa **Caixa de Estado** são do tipo ***Moore*** – **dependem apenas do estado**
- Transferências e saídas numa **Caixa de Saída Condicional** são do tipo ***Mealy*** – **dependem tanto do estado quanto das entradas**

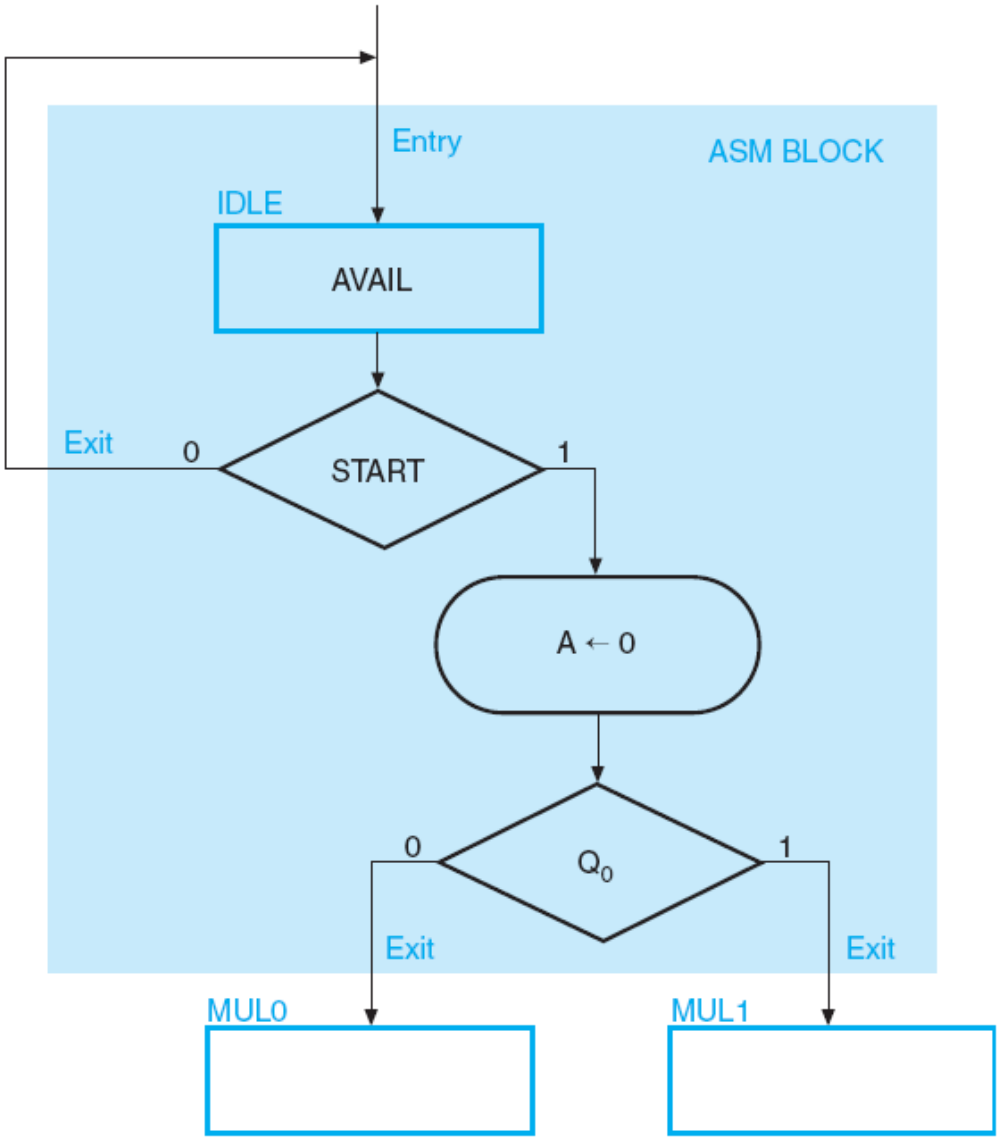
Como Conectar Caixas

- Ao conectar as caixas, é possível ver a conveniência desta representação
- Quais são:
  - Transferências?
  - As saídas?
  - As entradas?
  - Transferências Condicionais?
  - As saídas condicionais?



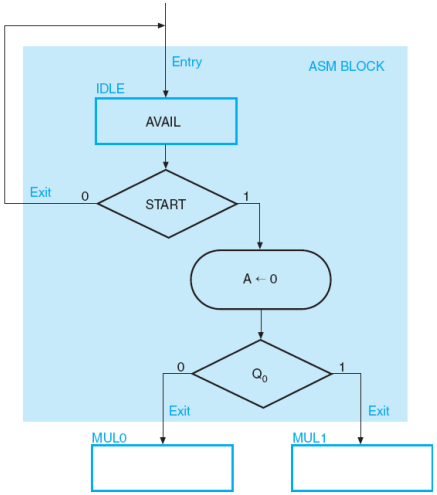
Bloco *ASM*

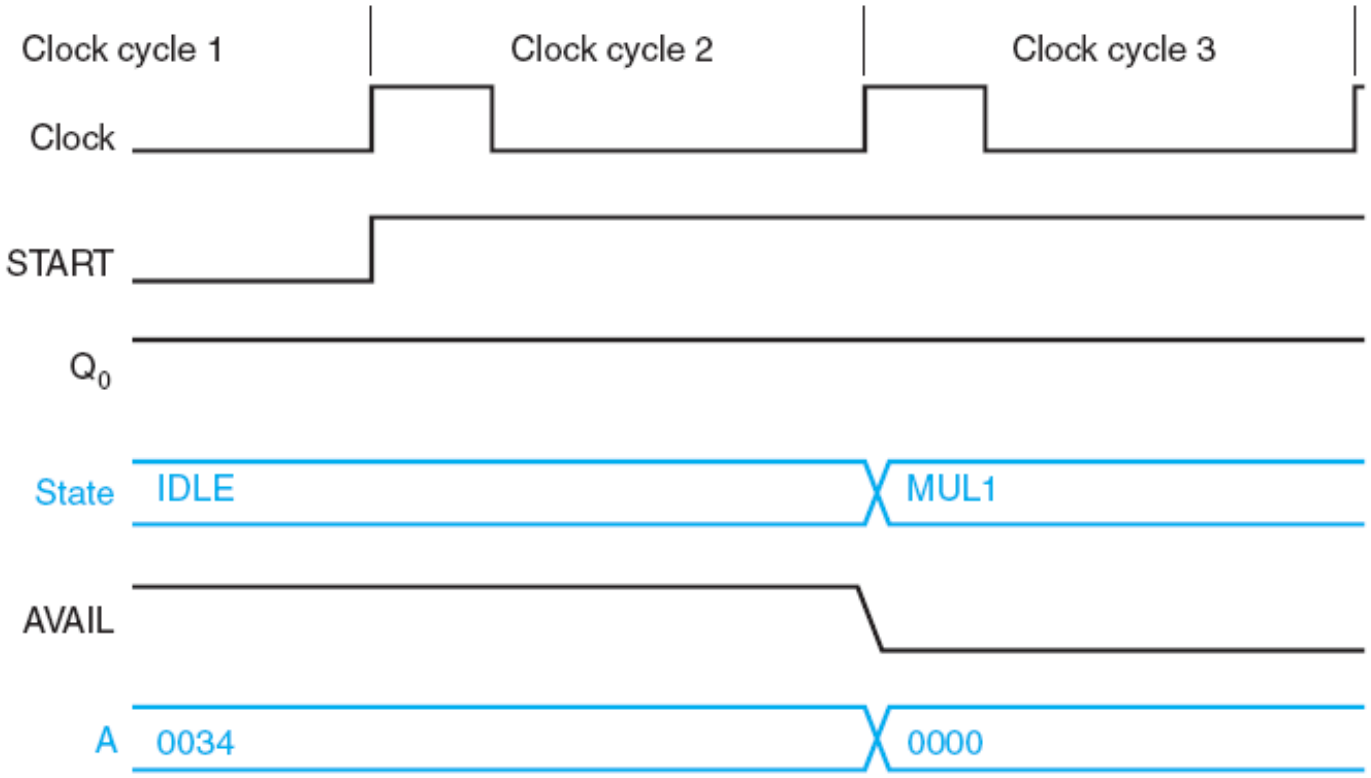
- Uma **Caixa de Estado** com todas as **Caixas de Decisão** e **Saída Condicional** conectadas a ela constitui um **Bloco *ASM***
- Um **Bloco *ASM*** inclui todos os itens do caminho e estado atual para ele mesmo ou outros estados, e representa as **ações de decisão ou saída** que podem ocorrer no estado indicado na entrada do bloco *(Entry)*



Tempos no Diagrama *ASM*

- As **Saídas** aparecem enquanto a máquina estiver no estado (ex. *AVAIL*)
- **Transferências Entre Registradores** ocorrem no *clock* de saída do estado – Novos valores ocorrem no novo estado!





Na borda de subida do *clock* no ciclo 2, ***START*** ainda está em ***0***, por isso ***AVAIL*** continua em ***1***

Mas na borda de subida do *clock* no ciclo 3, ***START*** está em ***1***, acarretando uma mudança de estado (***IDLE*** → ***MUL1***), e a consequente realização da operação **A ← 0**

Exemplo de Multiplicação Manual

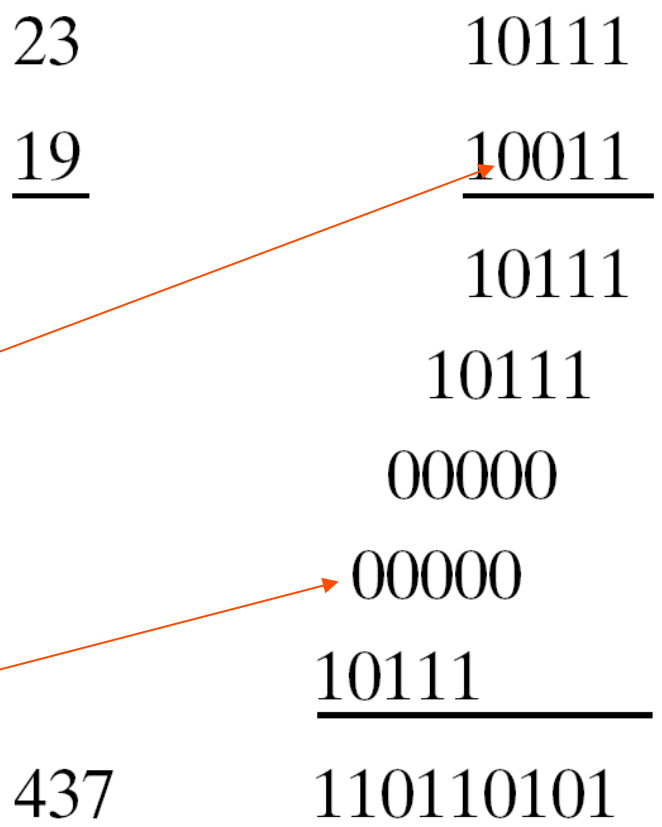
- Exemplo: $(10111 \times 10011)_2$
- Note que a soma dos produtos parciais de *n* dígitos requer a soma de até *n* dígitos (com *vai-um*) em cada coluna
- Note também que uma multiplicação de *n* **x** *m* dígitos gera um resultado de até *m* + *n* dígitos

Como implementar este algoritmo em ***hardware* digital**?

23	Multiplicando	10111
19	Multiplicador	10011
		10111
		10111
Produtos Parciais		00000
		00000
		10111
437	Produto	110110101

Modificações no Algoritmo Manual

- Em vez de usar um somador que some ***n* números** simultaneamente, é mais barato ter um circuito que some **dois números** apenas
- Em vez de fazer um ***shift* para a esquerda do multiplicando** a ser somado ao produto parcial, é melhor fazer um ***shift* para a direita do produto parcial** (assim, é possível usar um somador com *n* posições apenas, em vez de um de *2n* posições)
- Quando o *bit* correspondente do multiplicador for *0*, não há necessidade de somar todos os *0´s* ao produto parcial, já que eles não alteram o resultado



Algoritmo de *Hardware*

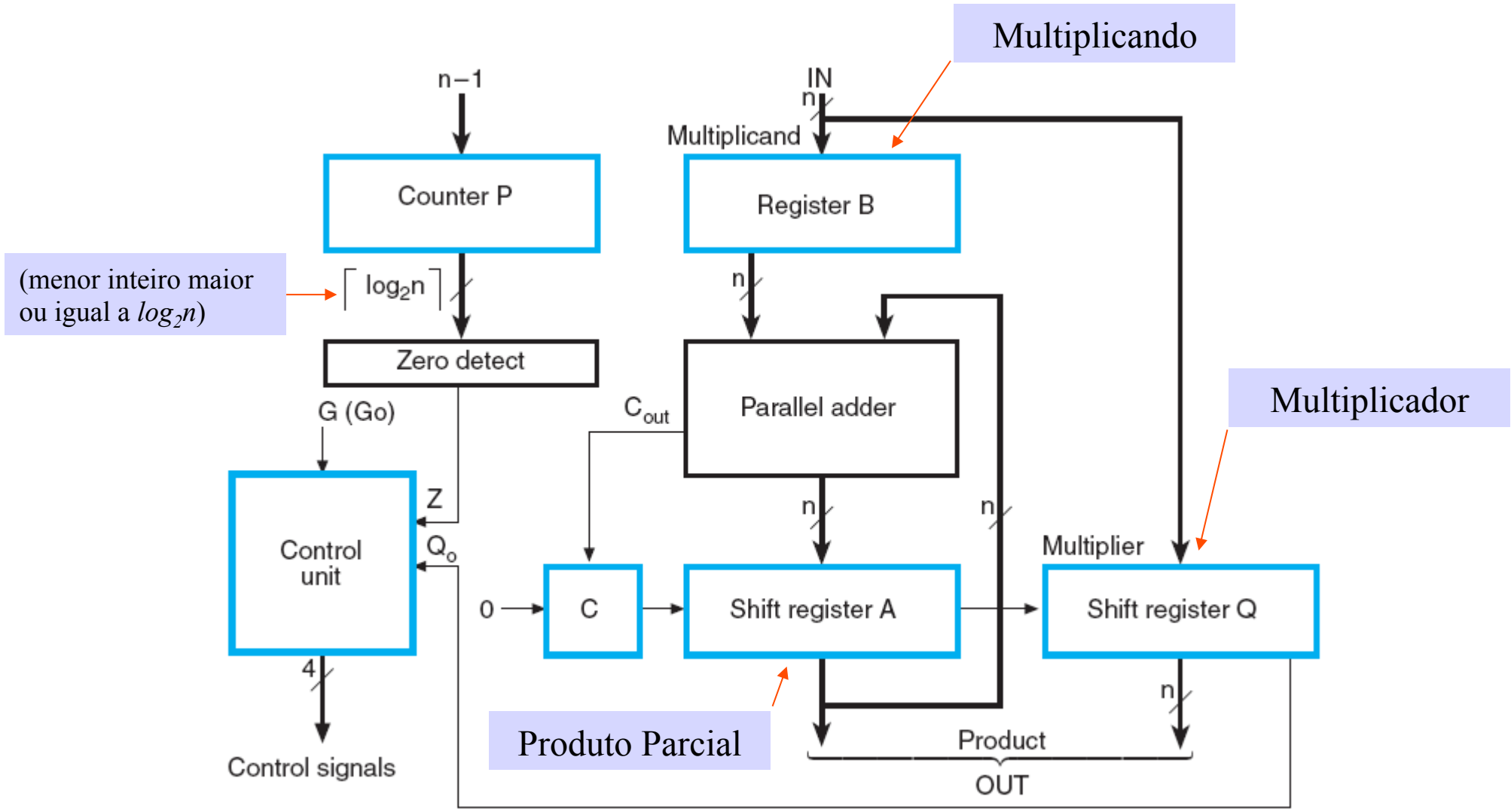
23	10111	Multiplicando
19	10011	Multiplicador
	00000	Produto parcial **inicial**
	10111	Como o *bit* multiplicador é *1*, some o multiplicando
	10111	Produto parcial depois da soma e antes do *shift*
	010111	Produto parcial depois do *shift*
	10111	Como o *bit* multiplicador é *1*, some o multiplicando
	1000101	Produto parcial depois da soma e antes do *shift*
	1000101	Produto parcial depois do *shift*
	01000101	Produto parcial depois do *shift* (multiplicador = 0)
	001000101	Produto parcial depois do *shift* (multiplicador = 0)
	10111	Como o *bit* multiplicador é *1*, some o multiplicando
	110110101	Produto parcial depois da soma e antes do *shift*
437	0110110101	Produto depois do *shift* final

Toda vez que o *bit* do multiplicador for *1*, é adicionado o multiplicando, seguido de um *shift* para a direita do produto parcial

Se o *bit* do multiplicador for *0*, somente o *shift* para a direita é efetuado

Overflow (*vai-um*)

Diagrama de Blocos para o Multiplicador



Operações no Multiplicador

1. O **multiplicando** (1º operando) é carregado no **registrador *B***
2. O **multiplicador** (2º operando) é carregado no **registrador *Q***
3. Os **registradores *C||A* são inicializados em *0*** quando ***G* recebe *1***
4. Os **produtos parciais** são formados nos **registradores *C||A||Q***
5. Cada *bit* do multiplicador, começando com o *LSB*, é processado (se o *bit* for *1*, usar o somador para somar *B* ao produto parcial; se o *bit* for 0, não fazer nada)
6. O conteúdo dos **registradores *C||A||Q*** são **deslocados para a direita**
   - Os *bits LSB* do produto parcial vão preenchendo as posições vacantes *MSB* de *Q* na medida em que o multiplicador vai sendo deslocado para fora de *Q*
   - Se ocorrer ***overflow*** na adição, o *vai-um* é recuperado do ***FF C*** durante o deslocamento para a direita
7. Os passos *5* e 6 são repetidos até que a condição *Counter P = 0* seja atingida e sinalizada pelo detector de zero (*Zero detect*)
   - O Contador *P* é inicializado no passo *4* em *n – 1*, n = nº. de *bits* no multiplicador, e seu conteúdo é testado antes de fazer um decremento

Diagrama *ASM* para o Multiplicador

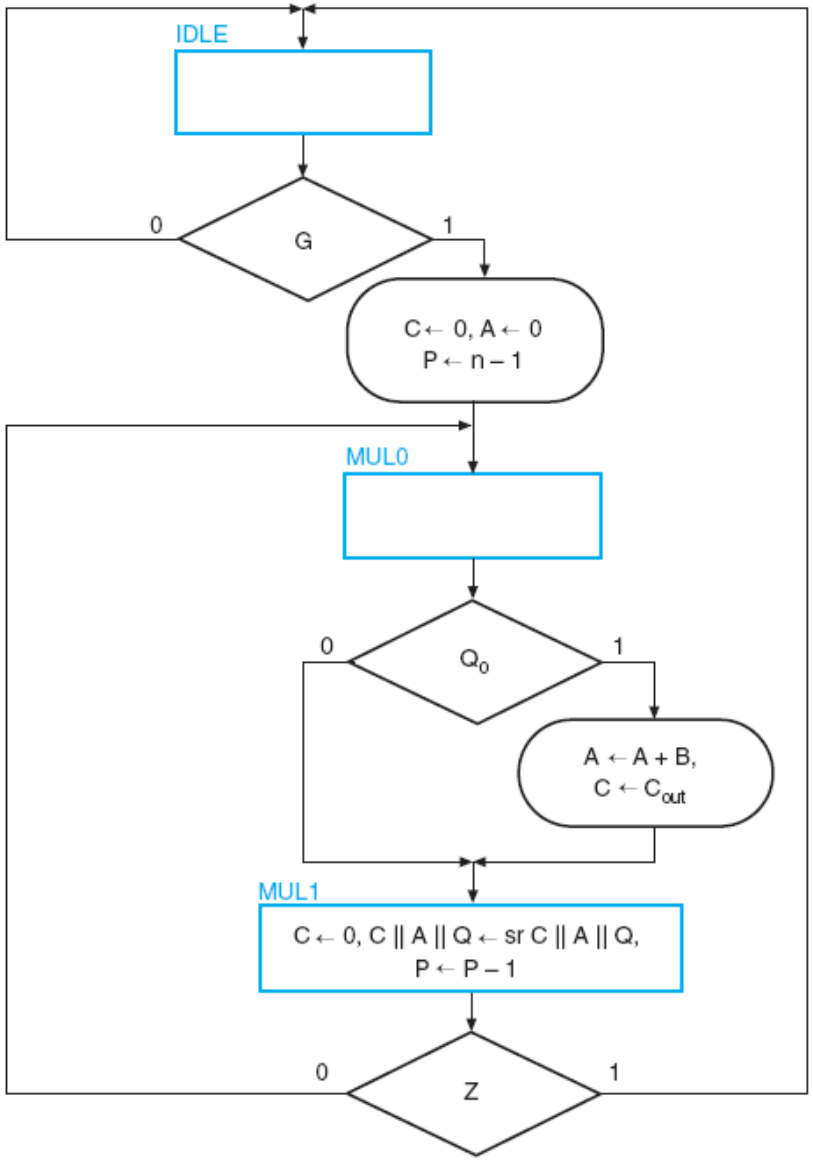
Inicialmente, o **multiplicando** é colocado em *B*, enquanto que o **multiplicador**, em *Q*. Esta operação de carregamento dos registradores não é feita pela **Unidade de Controle do Multiplicador**

Quando *G* recebe *1*, autorizando o início da multiplicação, os registradores *C* e *A* são carregados com zeros, enquanto que o contador *P*, com *n-1*

No estado *MUL0*, é tomada uma decisão baseada em Q_0, o *bit LSB* de *Q*. Se Q_0 for *1*, o conteúdo de *B* é somado ao de *A*, e o valor de *Cout* (*vai-um*) é carregado no *FF C*. Se for *0, A* e *C* ficam inalterados

No estado *MUL1*, é realizado um deslocamento para a direita nos conteúdos de *C, A* e *Q*, enquanto que o contador *P* é decrementado em *1*

A sucessão de estados *MUL0* e *MUL1* continua até que o contador *P* chegue a *0* e *Z* assuma o valor *1*



Considerações Sobre o Diagrama *ASM*

- Três estados foram usados de acordo com um modelo combinado *Mealy - Moore*:
  - **Estado *IDLE*** – no qual:
    - as saídas da multiplicação anterior são mantidas até que *Q* seja carregado com o novo multiplicando
    - a entrada *G* é usada como condição para iniciar a multiplicação, e
    - *C, A* e *P* são inicializados
  - **Estado *MUL0*** – no qual:
    - a adição condicional é realizada baseada no valor de Q_0
  - **Estado *MUL1*** – no qual:
    - é realizado um deslocamento para a direita para obter o produto parcial e posicionar o próximo *bit* do multiplicador em Q_0
    - é verificada a contagem final até *0*, realizada no contador *P*, para avaliar o término ou a continuação da multiplicação

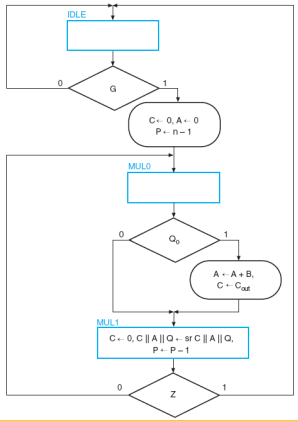
Controle por *Hardware*

- Na implementação da **Unidade de Controle** há dois aspectos distintos:
  - O **controle das microoperações** – responsável por gerar os sinais de controle que vão acionar os registradores, o somador etc. no **Caminho de Dados**
  - O **sequenciamento das microoperações através da Unidade de Controle** – responsável por determinar o que fazer depois de cada microoperação

- Por isso, o Diagrama ASM original será dividido em dois aspectos distintos:
  - Uma **Tabela de Sinais de Controle**, que define os sinais de controle em termos de **estados** e **entradas**
  - Um **Diagrama ASM Simplificado**, que apresenta apenas as transições de estado para estado

- Embora estes dois aspectos estejam sendo separados por razões de projeto, eles podem compartilhar a mesma lógica

Tabela de Sinais de Controle



Block Diagram Module	Microoperation	Control Signal Name	Control Expression
Register A:	$A \leftarrow 0$	Initialize	$IDLE \cdot G$
	$A \leftarrow A + B$	Load	$MUL0 \cdot Q_0$
	$C \| A \| Q \leftarrow \text{sr } C \| A \| Q$	Shift_dec	MUL1
Register B:	$B \leftarrow IN$	Load_B	LOADB
Flip-Flop C:	$C \leftarrow 0$	Clear_C	$IDLE \cdot G + MUL1$
	$C \leftarrow C_{out}$	Load	—
Register Q:	$Q \leftarrow IN$	Load_Q	LOADQ
	$C \| A \| Q \leftarrow \text{sr } C \| A \| Q$	Shift_dec	—
Counter P:	$P \leftarrow n - 1$	Initialize	—
	$P \leftarrow P - 1$	Shift_dec	—

Os **Sinais de Controle** necessários para o Caminho de Dados foram baseados no Diagrama ASM do Multiplicador

O nome de um estado pode ser tratado como uma variável booleana

- Com base no **ASM do Multiplicador** foram examinados os **registradores do Caminho de Dados**, tabuladas as microoperações para cada registrador e definidos os respectivos **Sinais de Controle**
- Um **Sinal de Controle** pode ser usado para ativar microoperações em mais de um registrador
- As expressões booleanas para cada Sinal de Controle são derivadas da localização da microoperação no **Diagrama ASM**. Por ex., para o **registrador *A*** há três microoperações no Diagrama ASM: (1) *clear*, (2) somar e carregar e (3) *right shift*

Tabela de Controle

- Os sinais foram definidos pressupondo-se operações com registradores
- ***LOADQ*** e ***LOADB*** são **sinais controlados externamente** ao sistema que utiliza o multiplicador e não serão tratados nesta parte do projeto
- Note que vários dos sinais de controle são "reutilizados" por diferentes registradores (por ex., ***Initialize*** é usado para ***inicializar* o registrador A** e **carregar o contador P**)
- Estes sinais de controle são as "***saídas***" da **Unidade de Controle**
- Tendo as saídas representadas na **Tabela de Sinais de Controle**, elas foram removidas do **Diagrama *ASM*** produzindo um *ASM* que representa apenas o comportamento sequencial, ou seja, determina qual será o próximo estado

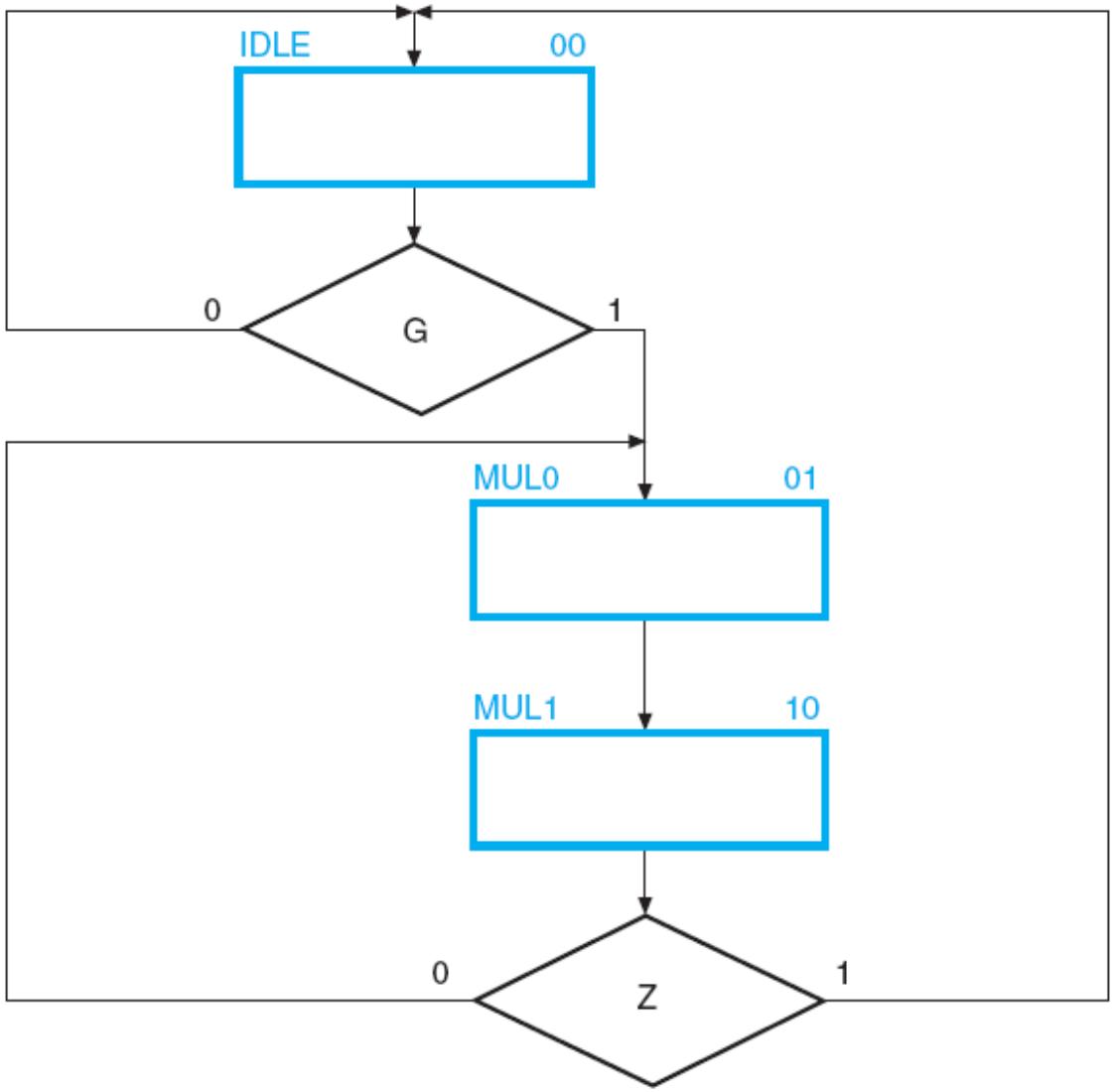
Parte do Sequenciamento do *ASM*

•Com a informação das microoperações removidas, o Diagrama *ASM* pode ser simplificado e refeito para ressaltar apenas o **sequenciamento das operações**

•Como todas as **Caixas de Saída Condicional** foram removidas, também as Caixas de Decisão que não afetam o Próximo Estado foram retiradas

•Agora a parte do **sequenciamento** da **Unidade de Controle**, i.e., o comportamento do **Próximo Estado** pode ser visualizado mais facilmente

•Devido a esta correspondência, este **Diagrama ASM Modificado** se assemelha a um **Diagrama de Estado**, e a partir dele pode-se construir uma **Tabela de Estados** para o sequenciamento da Unidade de Controle



Logic and Computer Design Fundamentals – Mano & Kime

Métodos de Projeto de Controle

- **Procedimento tradicional** - factível para os casos em que o número de estados é pequeno:
  - a partir de um **Diagrama de Estados**, constrói-se uma **Tabela de Estados**, códigos binários são atribuídos aos estados, e as **Equações de Entrada** dos *FFs* são deduzidas a partir dos valores do Próximo Estado da Tabela de Estados codificada
- **Procedimentos especializados** que usam um único sinal para representar cada estado:
  - **Procedimento do Registrador de Sequência e Decodificador**
    - Registrador de Sequência com estados codificados, ex., 00, 01, 10, 11
    - A saída do Decodificador produz sinais de "estado", ex., 0001, 0010, 0100, 1000 (em vez de usar as saídas dos *FFs*, um único sinal – a saída do Decodificador – indica em que estado a *MEF* está)
  - **Procedimento usando um *Flip-Flop* por Estado**
    - As saídas dos *FFs* são os sinais de "estado", ex., 0001, 0010, 0100, 1000 (a cada instante, apenas um *FF* possui saída Q=1)

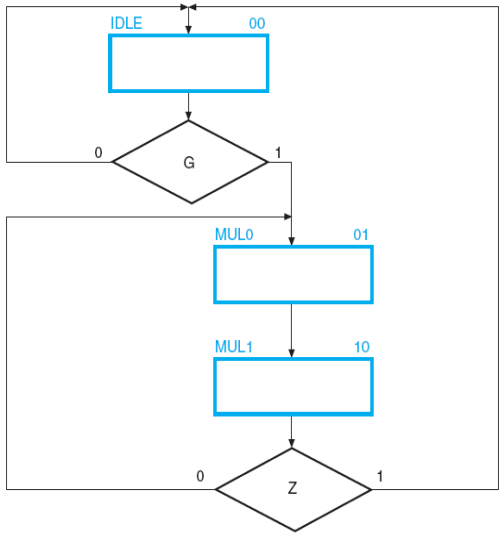
Projeto do Sequenciador e do Decodificador

- Inicialmente, usar as técnicas de projeto tradicionais
- Primeiramente, definir:
  - Estados: *IDLE*, *MUL0*, *MUL1*
  - Sinais de Entrada: *G, Z,* Q_0 (Q_0 afeta as saídas, não o próximo estado)
  - Sinais de Saída: *Initialize, LOAD, Shift_Dec, Clear_C*
  - Diagrama de Transição de Estado (usar o *ASM* do *slide* 25)
  - Função de Saída (usar a Tabela de Sinais de Controle do *slide* 23)
- A seguir, encontrar
  - Codificação de Estados (00, 01 ..)
  - Serão usados dois *bits* de estado para codificar os três estados *IDLE*, *MUL0* e *MUL1*

Estados	M1	M0
IDLE	0	0
MUL0	0	1
MUL1	1	0
Sem uso	1	1

Sequenciador e Decodificador

- Assumindo que as variáveis de estado *M1* e *M0* são decodificadas em estados (Estado Atual), o **Próximo Estado** da tabela é:

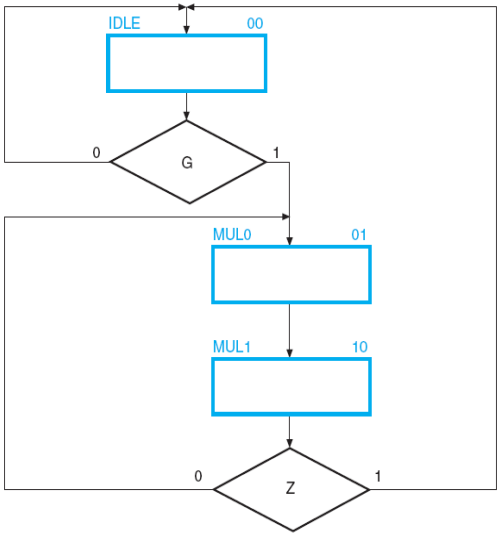


Estado Atual	Entr. G Z	Prox. Est. M1 M0
IDLE (00)	0 0	0 0
IDLE (00)	0 1	0 0
IDLE (00)	1 0	0 1
IDLE (00)	1 1	0 1
MUL0 (01)	0 0	1 0
MUL0 (01)	0 1	1 0
MUL0 (01)	1 0	1 0
MUL0 (01)	1 1	1 0

M1

Estado Atual M1 M0	Entr. G Z	Prox. Est. M1 M0
MUL1 (10)	0 0	0 1
MUL1 (10)	0 1	0 0
MUL1 (10)	1 0	0 1
MUL1 (10)	1 1	0 0
Sem uso	0 0	X X
Sem uso	0 1	X X
Sem uso	1 0	X X
Sem uso	1 1	X X

Tabela de Estados Simplificada para o Registrador de Sequência e o Decodificador



Present state			Inputs		Next state		Decoder Outputs		
Name	M_1	M_0	G	Z	M_1	M_0	IDLE	MUL0	MUL1
IDLE	0	0	0	×	0	0	1	0	0
	0	0	1	×	0	1	1	0	0
MUL0	0	1	×	×	1	0	0	1	0
MUL1	1	0	×	0	0	1	0	0	1
	1	0	×	1	0	0	0	0	1
—	1	1	×	×	×	×	×	×	×

•Em vez de usar as saídas dos *FFs* para representar as condições do Estado Atual, agora podemos usar as saídas do Decodificador para codificar esta mesma informação

Sequenciador e Decodificador (cont.)

- Encontrar as equações do **Próximo Estado** para *M1* e *M0* fica mais fácil porque os estados decodificados são referenciados pelos nomes na Tabela de Estados:

  $M1 = MUL0$

  $M0 = IDLE \cdot G + MUL1 \cdot \overline{Z}$

- Note que, como há cinco variáveis (*IDLE, MUL0, MUL1*, G e Z), é difícil usar um mapa de *Karnaugh*, por isso as equações reduzidas foram obtidas diretamente da Tabela de Estados
- As **Equações de Saída**, usando os estados decodificados, são:

  $Initialize = IDLE \cdot G$

  $Load = MUL0 \cdot Q_0$

  $Clear\_C = IDLE \cdot G + MUL1$

  $Shift\_dec = MUL1$

  – (consultar a Tabela de Sinais de Controle do *slide* 23)

Present state			Inputs		Next state		Decoder Outputs		
Name	M_1	M_0	G	Z	M_1	M_0	IDLE	MUL0	MUL1
IDLE	0	0	0	×	0	0	1	0	0
	0	0	1	×	0	1	1	0	0
MUL0	0	1	×	×	1	0	0	1	0
MUL1	1	0	×	0	0	1	0	0	1
	1	0	×	1	0	0	0	0	1
—	1	1	×	×	×	×	×	×	×

Sequenciador e Decodificador (cont.)

- Fazendo otimização de nível múltiplo, definir $START = IDLE \cdot G$:

  $START = IDLE \cdot G$

  $M1 = MUL0$

  $M0 = START + MUL1 \cdot \overline{Z}$

  $Initialize = START$

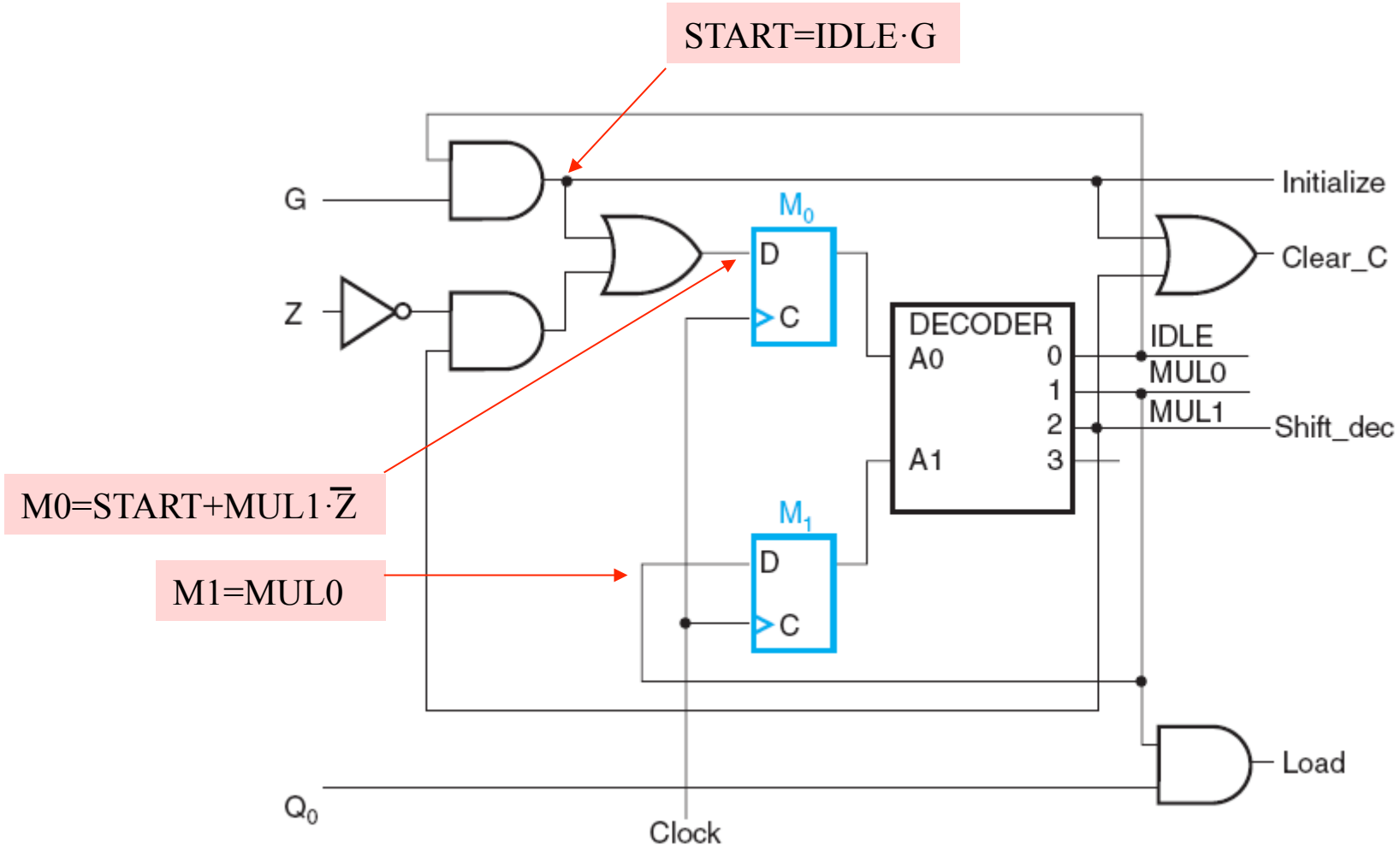
  $Load = MUL0 \cdot Q_0$

  $Clear\_C = START + MUL1$

  $Shift\_dec = MUL1$

- O circuito resultante usando *flip-flops D*, um decodificador, e as equações acima são apresentadas no próximo *slide*

Unidade de Controle Usando um Registrador de Sequência e um Decodificador

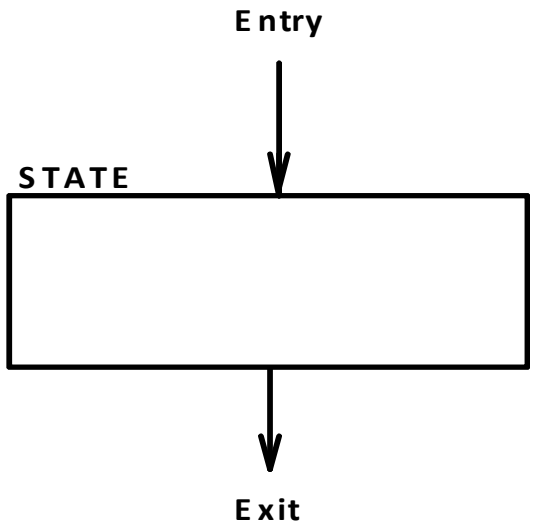


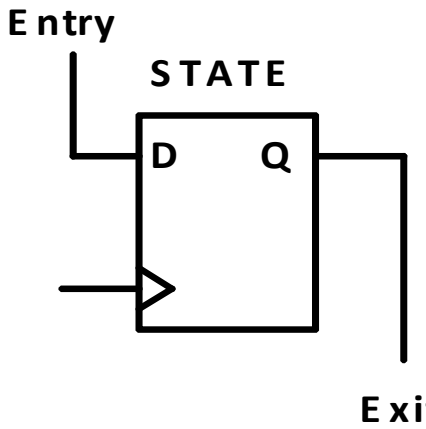
Procedimento Alternativo: Um *Flip-Flop* por Estado

- Este método usa **um *flip-flop* por estado** e um conjunto simples de regras de transformação para implementar o circuito
- O projeto começa com o **Diagrama *ASM*** e substitui
  1. **Caixas de Estado** por ***Flip-Flops***,
  2. **Caixas de Decisão Escalar** por um **Demultiplexador** com 2 saídas,
  3. **Caixas de Decisão Vetorial** por um **Demultiplexador**
  4. **Junções** por uma **Porta *OR***, e
  5. **Saídas Condicionais** por **Portas *AND***

Regras de Transformação: Caixa de Estado

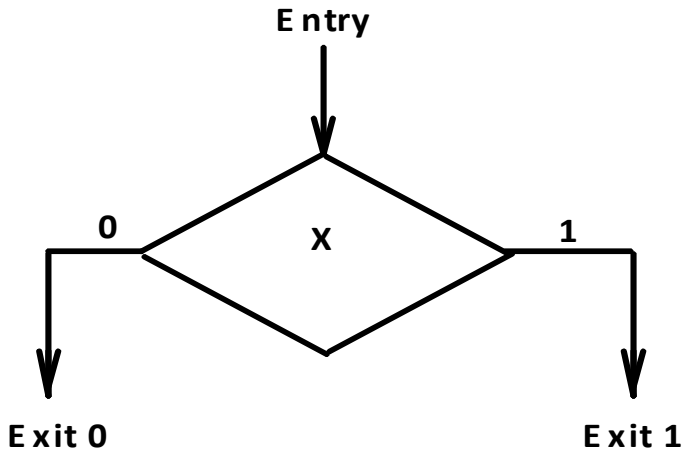
- Cada **Caixa de Estado** se transforma num ***Flip-Flop D***
- A entrada da Caixa de Estado é conectada à entrada *D* do *FF*
- A saída é conectada à saída *Q* do *FF*

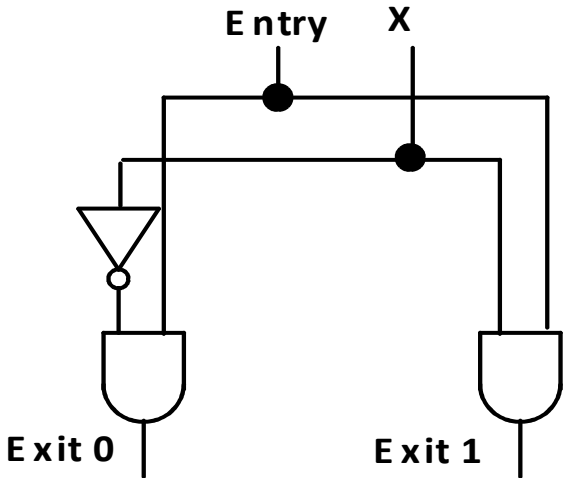




Regras de Transformação: Caixa de Decisão Escalar

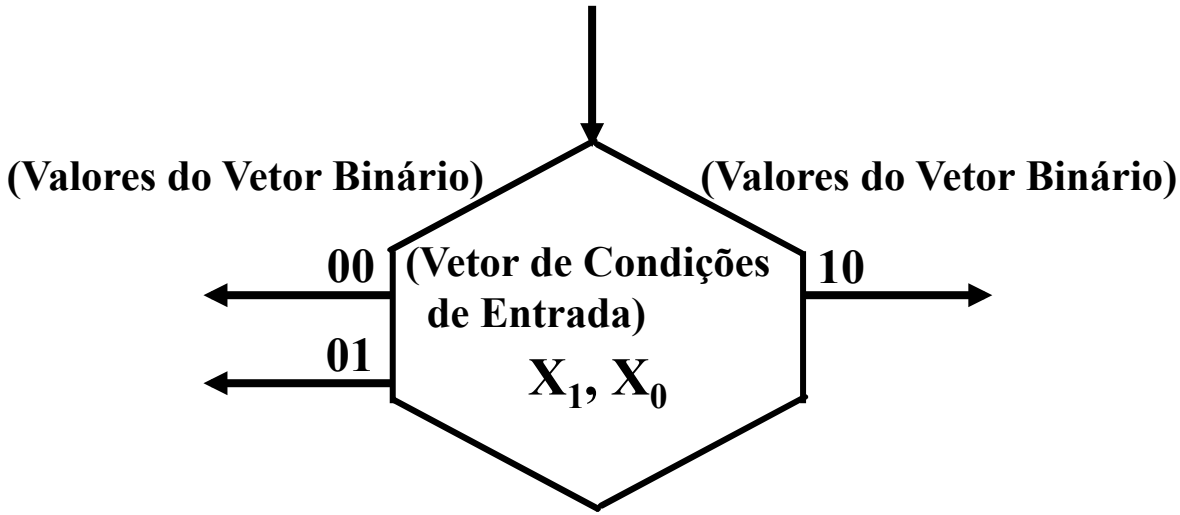
- Cada **Caixa de Decisão** se transforma num **Demultiplexador**
- A entrada (*Entry*) é a entrada habilitadora (*Enable inputs*)
- A Condição (*X*) é a Entrada de Seleção (*Select input*)
- As saídas decodificadas são os pontos de saída (*Exit points*)

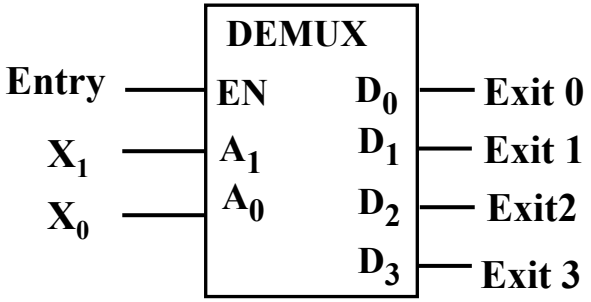




Regras de Transformação: Caixa de Decisão Vetorial

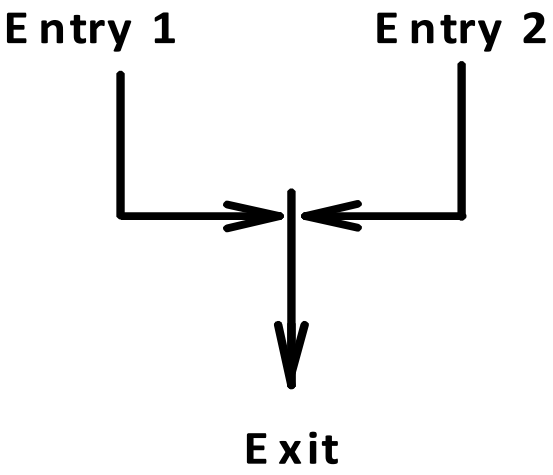
- Cada **Caixa de Decisão** se transforma num **Demultiplexador**
- A entrada (*Entry*) é a entrada habilitadora (*Enable input*)
- As Condições (..., X_1, X_0) são as entradas de seleção (*Select inputs*)
- As saídas do Demultiplexador são os pontos de saída (*Exit points*)

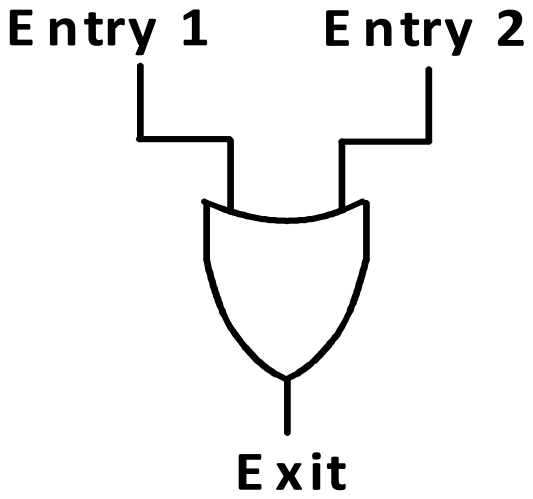




Regras de Transformação: Junção

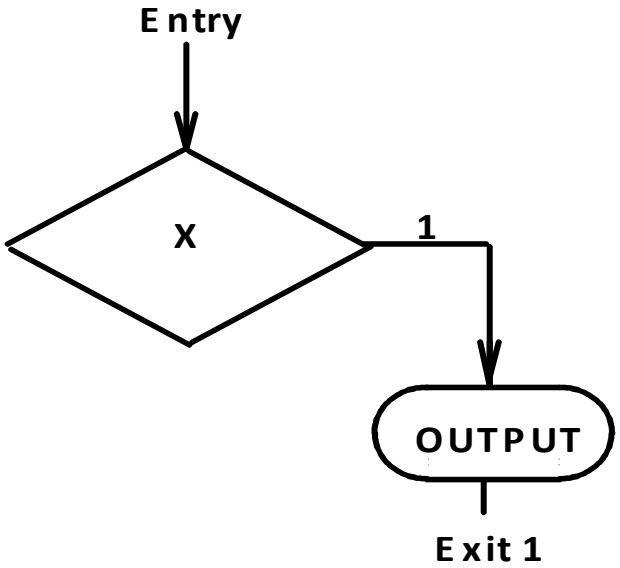
- Onde duas ou mais entradas se encontram, conectar as **variáveis de entrada** a uma porta ***OR***
- A Saída (*Exit*) é a saída da porta *OR*

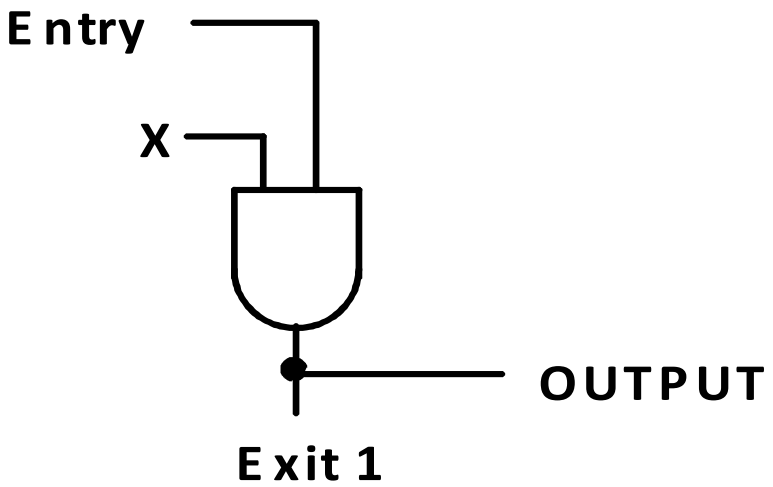




Regras de Transformação: Caixa de Saída Condicional

- A **entrada** é a **entrada de habilitação** (*Enable input*)
- A **Condição** (*X*) é a **entrada de seleção** (*Select input*)
- As saídas do Demultiplexador são os pontos de saída (*Exit points*)
- A saída de CONTROLE (*OUTPUT*) é o mesmo sinal do valor de saída (*Exit Value*)





Resumo das Regras de Transformação

Com estas regras de transformação, um *flip-flop* é atribuído a cada um dos estados, e, em qualquer instante, somente um dos *FFs* contém saída *1*, com todos os *FFs* restantes mantendo *0*

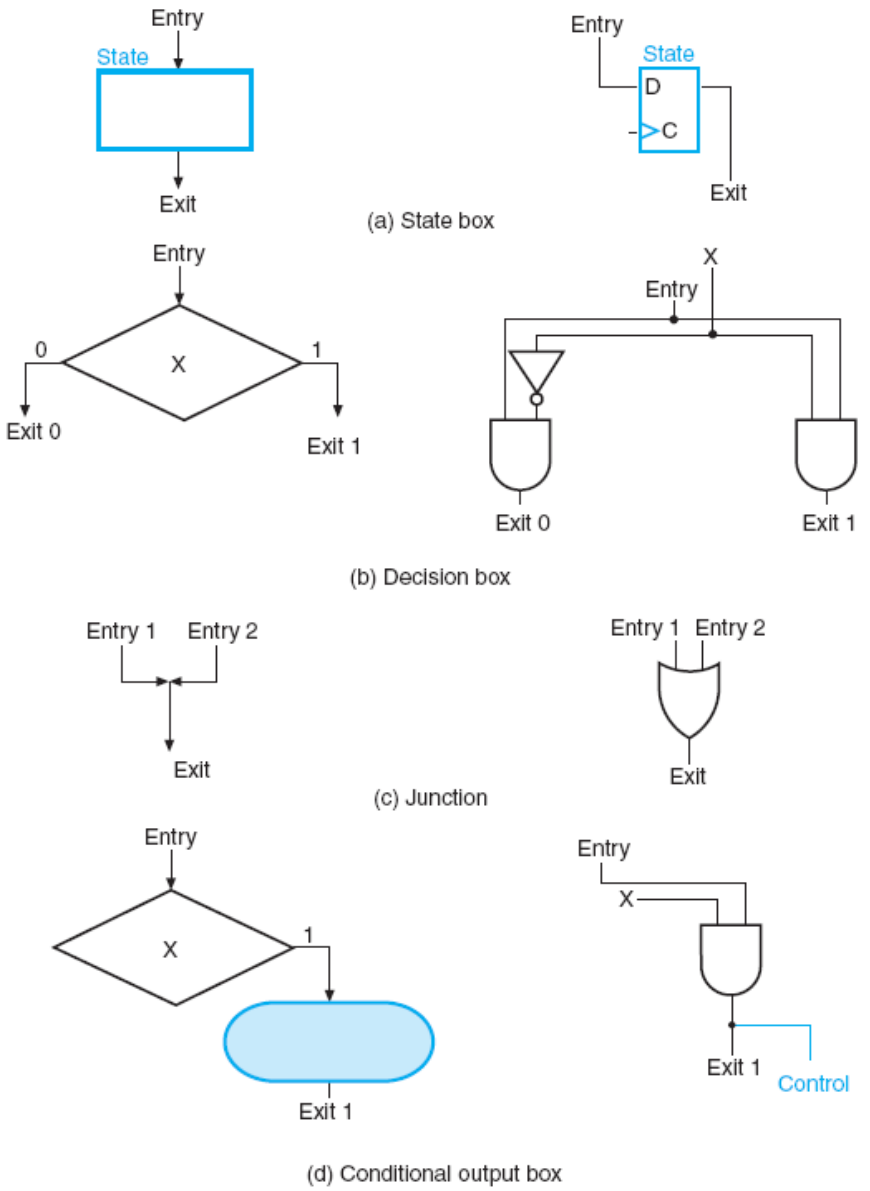
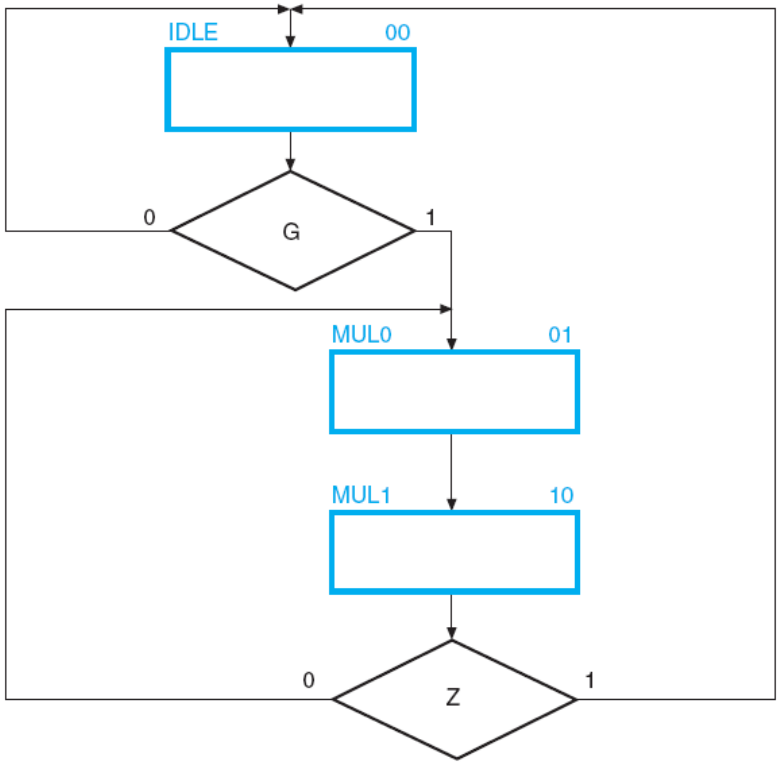
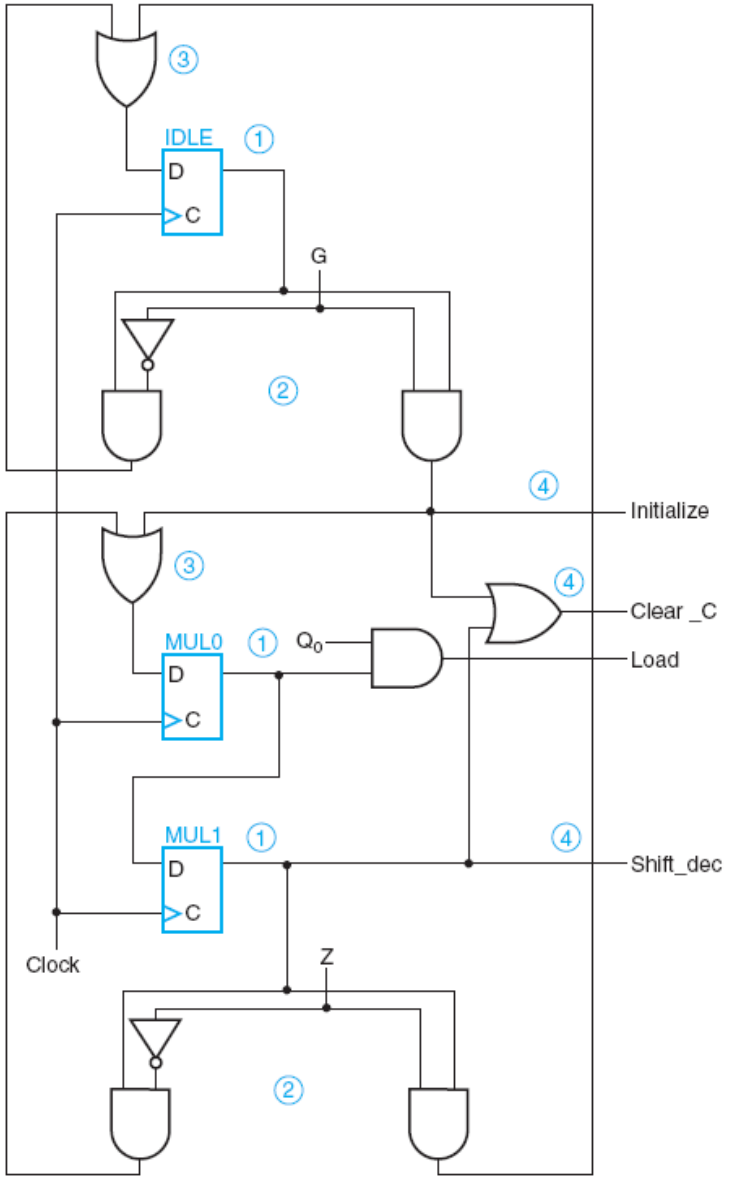


Diagrama Lógico da Unidade de Controle do Multiplicador com um *Flip-flop* por Estado





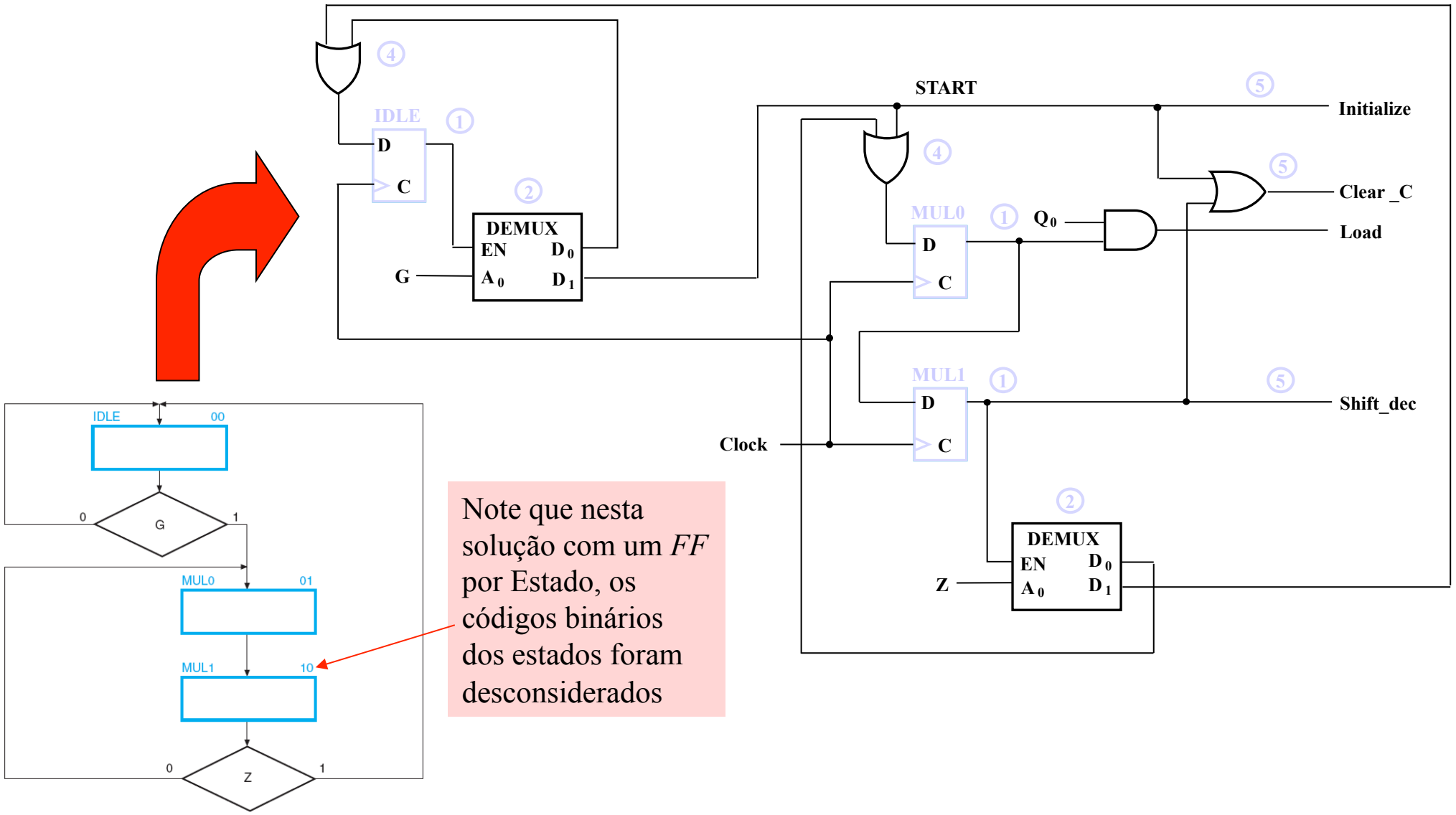
(1) Cada **Caixa de Estado** foi substituída por um ***FF* tipo *D***

(2) Cada **Caixa de Decisão** foi substituída por um **Demultiplexador**

(3) Cada Junção foi substituída por uma **porta *OR***

(4) As entradas e saídas das caixas do Diagrama ASM foram transportadas para as entradas e saídas dos componentes

Unidade de Controle com um *Flip-flop* por Estado para o Multiplicador Binário



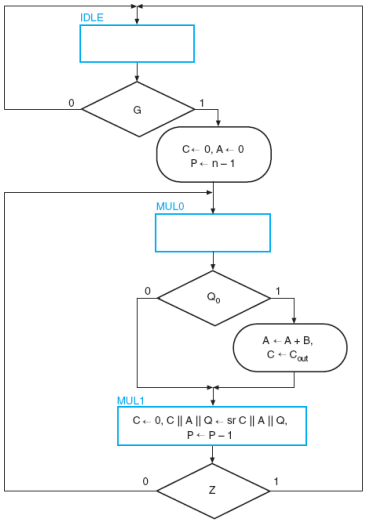
Note que nesta solução com um *FF* por Estado, os códigos binários dos estados foram desconsiderados

Descrição VHDL Comportamental do Multiplicador de 4 *bits* de Entrada

```
-- Binary Multiplier with n = 4: VHDL Description
-- See Figures 8-6 and 8-7 for block diagram and ASM Chart
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity binary_multiplier is
  port(CLK, RESET, G, LOADB, LOADQ: in std_logic;
    MULT_IN: in std_logic_vector(3 downto 0);
    MULT_OUT: out std_logic_vector(7 downto 0));
end binary_multiplier;

architecture behavior_4 of binary_multiplier is
  type state_type is (IDLE, MUL0, MUL1);
  signal state, next_state : state_type;
  signal A, B, Q: std_logic_vector(3 downto 0);
  signal P: std_logic_vector(1 downto 0);
  signal C, Z: std_logic;
begin
  Z <= P(1) NOR P(0);
  MULT_OUT <= A & Q;

  state_register: process (CLK, RESET)
  begin
    if (RESET = '1') then
      state <= IDLE;
    elsif (CLK'event and CLK = '1') then
      state <= next_state;
    end if;
  end process;
```




```
next_state_func: process (G, Z, state)
begin
  case state is
    when IDLE =>
      if G = '1' then
        next_state <= MUL0;
      else
        next_state <= IDLE;
      end if;
    when MUL0 =>
      next_state <= MUL1;
    when MUL1 =>
      if Z = '1' then
        next_state <= IDLE;
      else
        next_state <= MUL0;
      end if;
  end case;
end process;
```

```
datapath_func: process (CLK)
variable CA: std_logic_vector(4 downto 0);
begin
  if (CLK'event and CLK = '1') then
    if LOADB = '1' then
      B <= MULT_IN;
    end if;
    if LOADQ = '1' then
      Q <= MULT_IN;
    end if;
    case state is
      when IDLE =>
        if G = '1' then
          C <= '0';
          A <= "0000";
          P <= "11";
        end if;
      when MUL0 =>
        if Q(0) = '1' then
          CA := ('0' & A) + ('0' & B);
        else
          CA := C & A;
        end if;
          C <= CA(4);
          A <= CA(3 downto 0);
      when MUL1 =>
        C <= '0';
        A <= C & A(3 downto 1);
        Q <= A(0) & Q(3 downto 1);
        P <= P - "01";
    end case;
  end if;
end process;
end behavior_4;
```

Observações sobre a Descrição VHDL Comportamental do Multiplicador

•No início da *architecture*, uma declaração de tipo define os três estados ***IDLE, MUL0*** e ***MUL1***

•Sinais internos são declarados a seguir (***state, next_state, A, B, P, Q*** e ***C***)

•O sinal intermediário ***Z*** é declarado apenas por conveniência; o comando de atribuição ***Z<=P(1) NOR P(0)*** força ***Z*** ao valor ***1*** quando a contagem ***P*** atingir ***00***

•As saídas concatenadas dos registradores ***A*** e ***Q*** são atribuídas à saída ***MULT_OUT***; isto é necessário, para permitir que ***A*** e ***Q*** sejam usadas dentro do circuito e não apenas como saídas do circuito

•Há três processos na descrição VHDL; no primeiro é modelado o **Registrador de Estado**, além do ***clock*** e do ***RESET***; o segundo processo descreve a função do **Próximo Estado**, contendo em sua lista de sensitividade todos os sinais que podem afetar o próximo estado (***G, Z*** e ***state***); o terceiro processo descreve a função do **Caminho de Dados** (o primeiro ***if*** controla o carregamento do **Multiplicando** no **Registrador *B*** e o segundo ***if*** controla o carregamento do **Multiplicador** no **Registrador *Q***)

•As transferências entre registradores diretamente envolvidos na multiplicação são controladas por um comando ***case***, dependente do **Estado de Controle**, da entrada ***G*** e dos sinais internos ***Q(0)*** e ***Z***

•No estado ***MUL0***, para realizar uma soma com vetores ***std_logic***, foi acrescentado nas declarações iniciais o pacote ***ieee.std_logic_unsigned.all***; para obter o ***vai_um***, foi acrescentado um ***0*** à esquerda de ***A*** e ***B*** para realizar uma soma com ***5 bits***; a seguir, o ***vai_um*** é guardado em ***CA***

Controle Microprogramado

- ***Controle Microprogramado*** — emprega uma Unidade de Controle com seus valores de controle binários armazenados como *palavras* (*words*) numa memória
- ***Microinstruções*** — cada palavra na Memória de Controle que especifica uma ou mais microoperações para o sistema
- ***Microprograma*** — uma sequência de microinstruções (que serão lidas e executadas sucessivamente)
- ***Memória de Controle*** — uma memória *RAM* ou *ROM* na Unidade de Controle guardando o microprograma ou as microinstruções
- ***Memória de Controle com Escrita*** *(Writable Control Memory)* — uma memória *RAM* na qual as microinstruções são carregadas durante o início de operações, tendo como fonte outra memória não-volátil (por ex., um *HD)* ou o console (teclado)

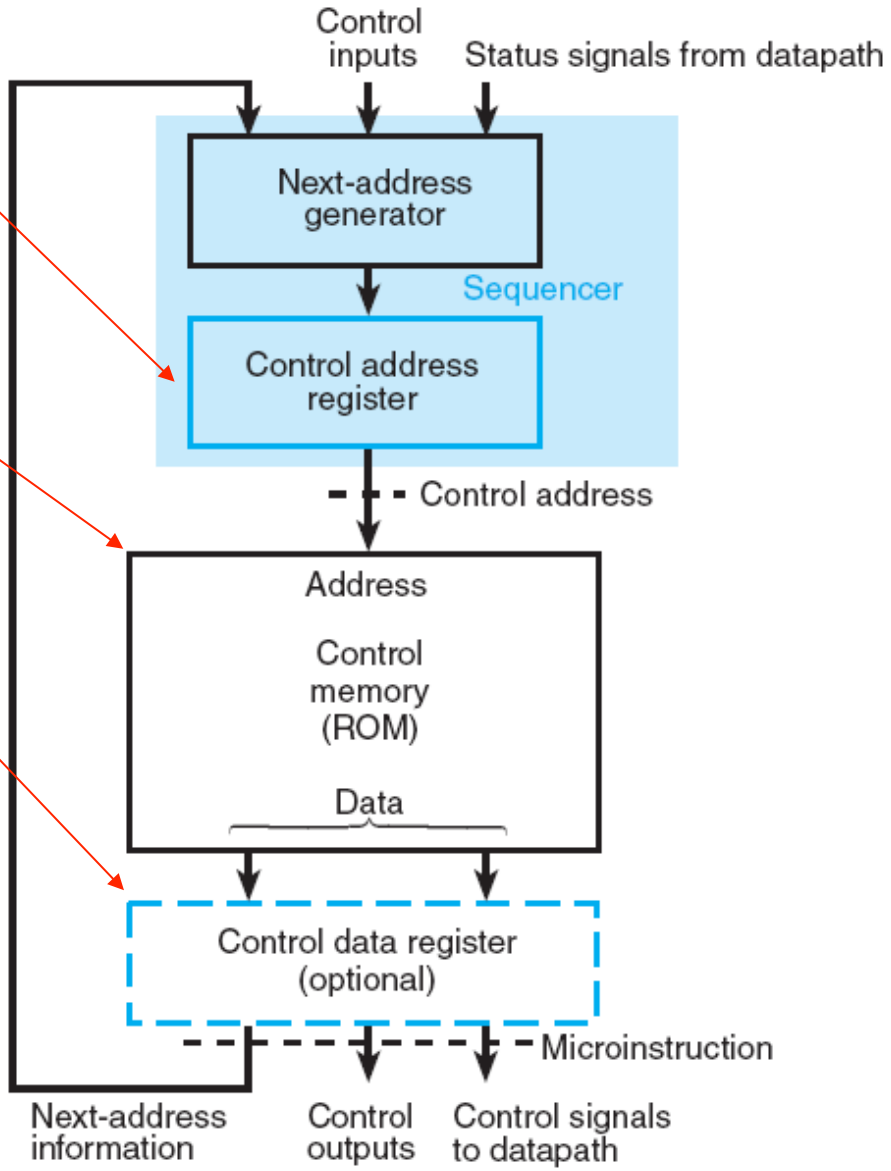
Organização da Unidade de Controle Microprogramada

•O **Registrador de Endereço de Controle** (***Control Address Register – CAR***) especifica o endereço da microinstrução a ser buscada na Memória de Controle

•A **Memória de Controle** pode ser uma ***ROM***, dentro da qual todas as informações de controle são armazenadas de forma permanente

•O **Registrador de Dado de Controle** (***Control Data Register – CDR***), que é opcional, guarda a microinstrução que está sendo executada pelo **Caminho de Dados** (*Datapath*) e pela **Unidade de Controle**

•Uma das funções da **Palavra de Controle** é determinar o endereço da próxima microinstrução a ser executada, que pode ser a instrução seguinte numa sequência ou estar armazenada em qualquer posição da **Memória de Controle**. Portanto, um ou mais *bits* que especifiquem como encontrar o endereço da próxima microinstrução deve(m) estar presente(s) na microinstrução atual

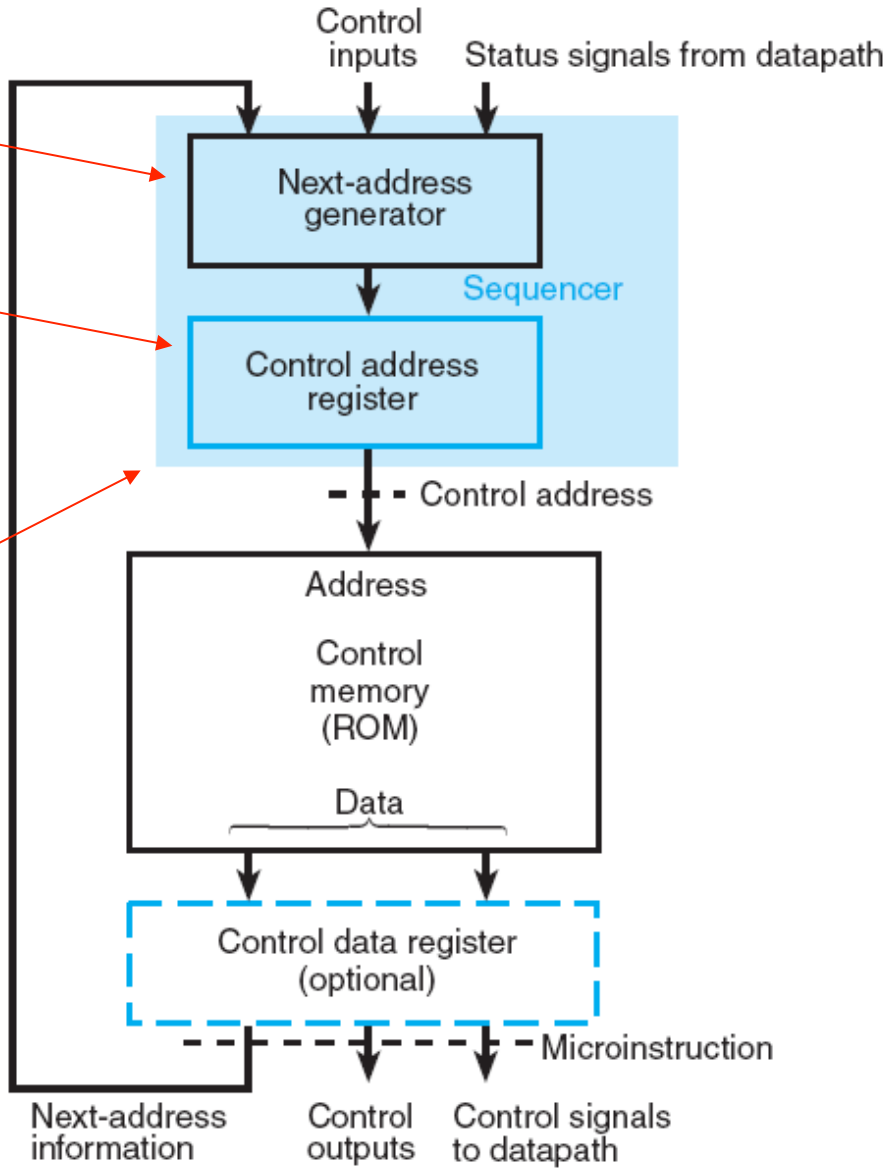


Sequenciador do Microprograma

•Enquanto uma microinstrução está sendo executada, o **Gerador de Próximo Endereço** (***Next-Address Generator***) produz o próximo endereço, que é transferido para o *CAR* no pulso seguinte de *clock* e é usado para ler a próxima microinstrução a ser executada a partir da ROM

•Dessa forma, as microinstruções contêm alguns *bits* para ativar as microoperações no **Caminho de Dados** (*Datapath*), além de outros *bits* que especificam a sequência de microinstruções que devem ser executadas

•O **Gerador de Próximo Endereço**, em combinação com o ***CAR***, às vezes, é chamado de **Sequenciador** do microprograma, já que ele determina a sequência de instruções que é lida da Memória de Controle



Alterações no Diagrama *ASM* da Unidade de Controle Microprogramada

•O endereço da próxima microinstrução pode ser especificada de diferentes formas, dependendo das entradas do **Sequenciador**. Por ex., o *CAR* pode ser incrementado pelo Sequenciador ou então ser carregado com um endereço obtido da Memória de Controle ou de uma fonte externa

•Os ***bits* de estado** (*status bits*) entram no **Gerador de Próximo Endereço** e afetam a determinação do Próximo Estado (representado pelo conteúdo do *CAR*). A menos que estes ***bits* de estado** contornem a Unidade de Controle e controlem diretamente as microoperações executadas no **Caminho de Dados**, a influência que eles podem exercer se resume a selecionar a próxima microoperação ao interferir no endereço gerado pelo Gerador de Próximo Endereço

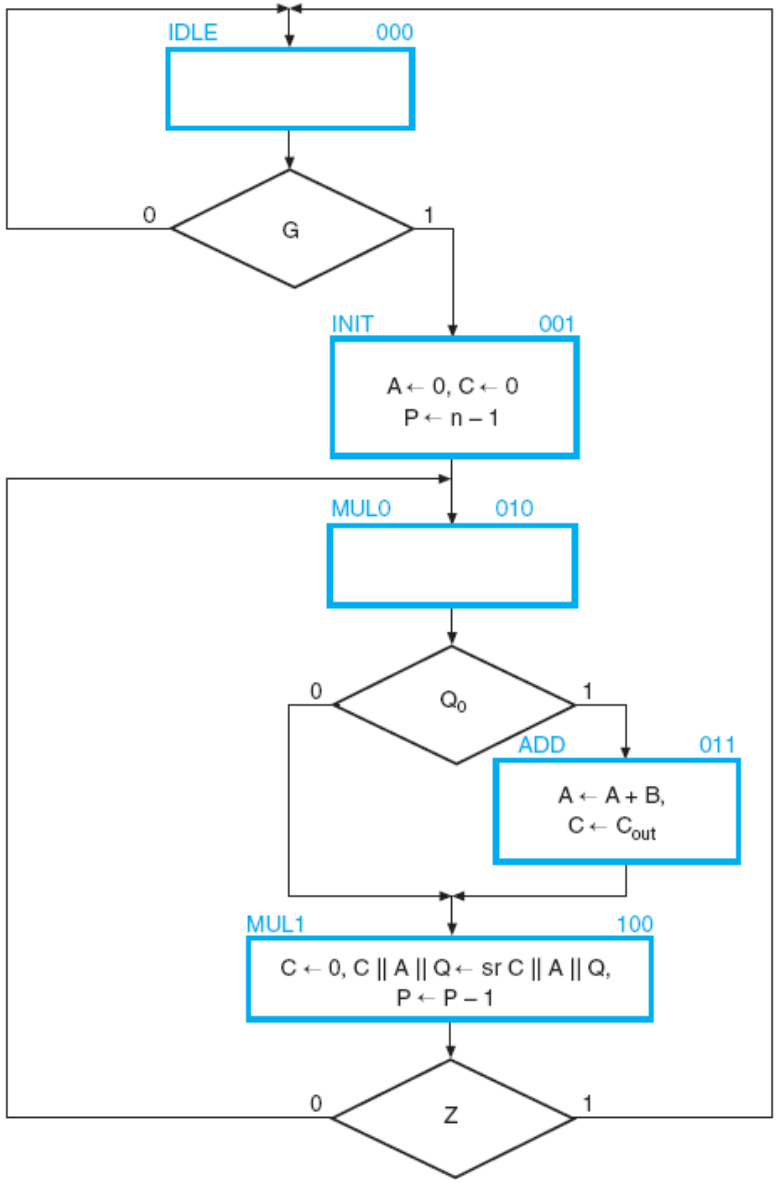
•Isto tem um efeito profundo na estrutura do Diagrama *ASM* para Controle Microprogramado, pois os Circuitos Sequenciais devem ser do tipo ***Moore*** e, consequentemente, não são permitidas **Caixas de Saídas Condicionais** no Diagrama *ASM*

•Isto significa mais estados serão necessários no Diagrama *ASM* para um dado algoritmo de *hardware*

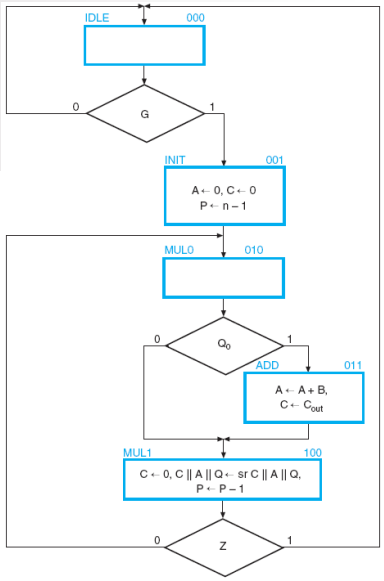
Diagrama *ASM* para a Unidade de Controle Microprogramada do Multiplicador Binário

O Diagrama *ASM* para o Multiplicador com Controle Microprogramado deve ter dois estados a mais (***INIT*** e ***ADD***), substituindo as duas Caixas de Saídas Condicionais do Diagrama *ASM* original

- Três parâmetros precisam ser determinados:
  - os *bits* na **Palavra de Controle** para as microinstruções
  - o tamanho da *ROM* e do *CAR*
  - a estrutura do Gerador de Próximo Endereço



Sinais de Controle para o Multiplicador Microprogramado



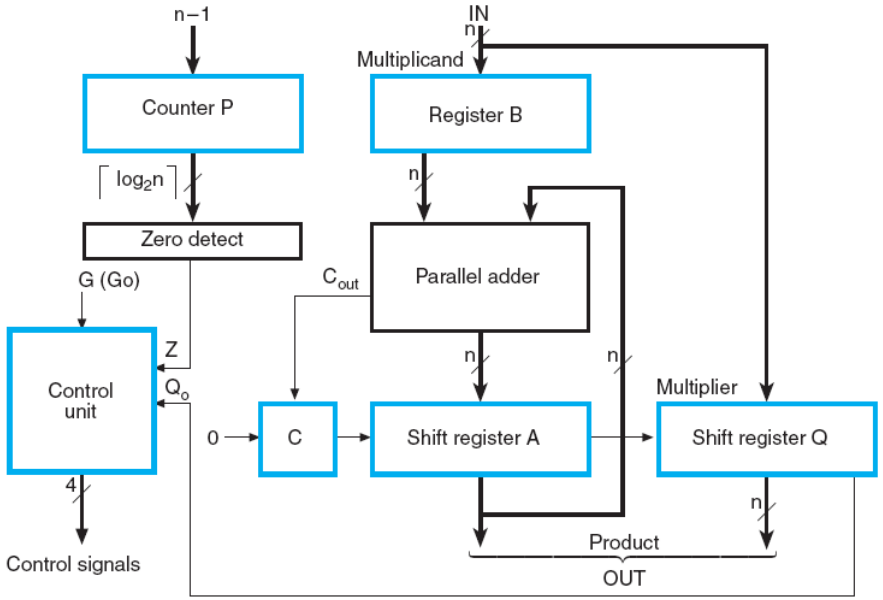
Apenas quatro **Sinais de Controle** são necessários para que o **Caminho de Dados** efetue uma multiplicação: *Initialize, Load, Clear_C* e *Shift_dec*

Control Signal	Register Transfers	States in Which Signal is Active	Micro-instruction Bit Position	Symbolic Notation
Initialize	$A \leftarrow 0, P \leftarrow n-1$	INIT	0	IT
Load	$A \leftarrow A+B, C \leftarrow C_{out}$	ADD	1	LD
Clear_C	$C \leftarrow 0$	INIT, MUL1	2	CC
Shift_dec	$C\|\|A\|\|Q \leftarrow \text{sr } C\|\|A\|\|Q, P \leftarrow P-1$	MUL1	3	SD

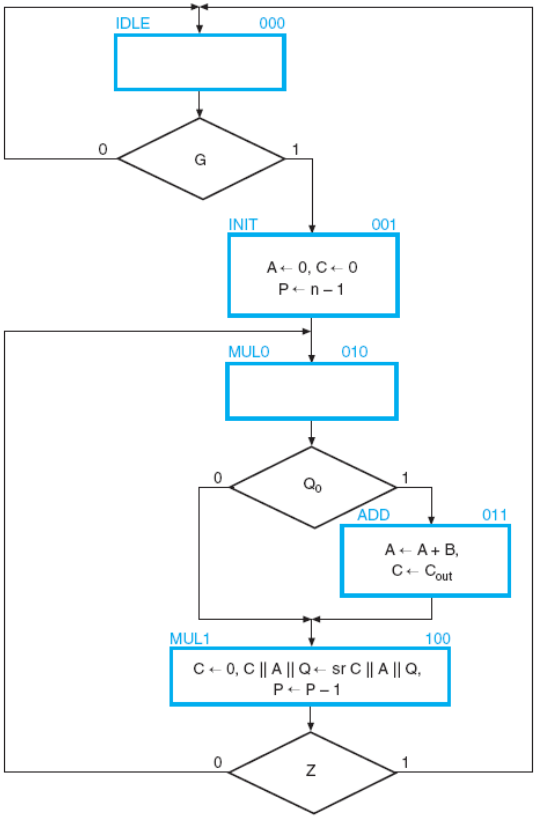
As denominações simbólicas das microoperações serão usadas na microprogramação

Com base no Diagrama *ASM* para o Controle Microprogramado, é possível determinar em quais **Estados** os quatro **Sinais de Controle** devem estar ativos. Esta informação é fundamental para projetar a parte da microinstrução que controla o **Caminho de Dados**

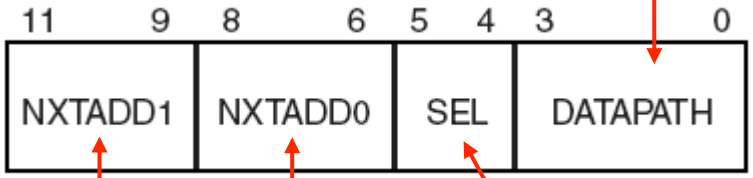
Os quatro Sinais de Controle serão representados por *bits* em quatro posições distintas na **Palavra de Controle**



Formato de Palavra de Controle para Microinstrução



Os quatro *bits* de código necessários para os quatro Sinais de Controle que atuam sobre o Caminho de Dados estão codificados no campo denominado *DATAPATH*

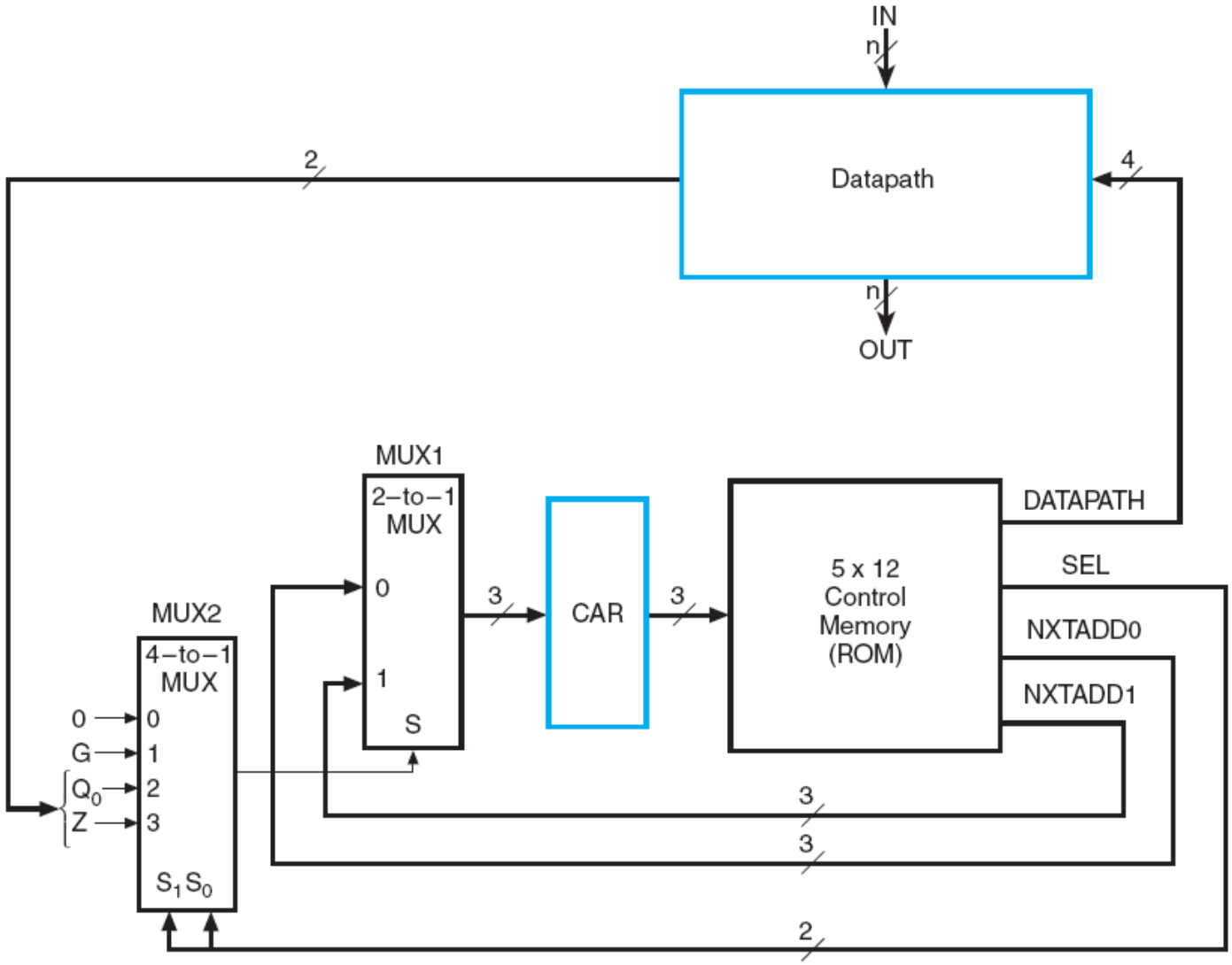


Para os casos em que o Próximo Estado depende de um ***bit* de estado** ou de um **valor de entrada**, é necessário um par de endereços, um para o valor de entrada *0* e outro para o valor de entrada *1*

A decisão sobre qual dos dois possíveis endereços usar será influenciada pelos ***bits* de estado** e **valores de entrada**, todos colocados nas entradas de um *MUX 4 x 1*, que é acionado pelos *2 bits* do campo *SEL*

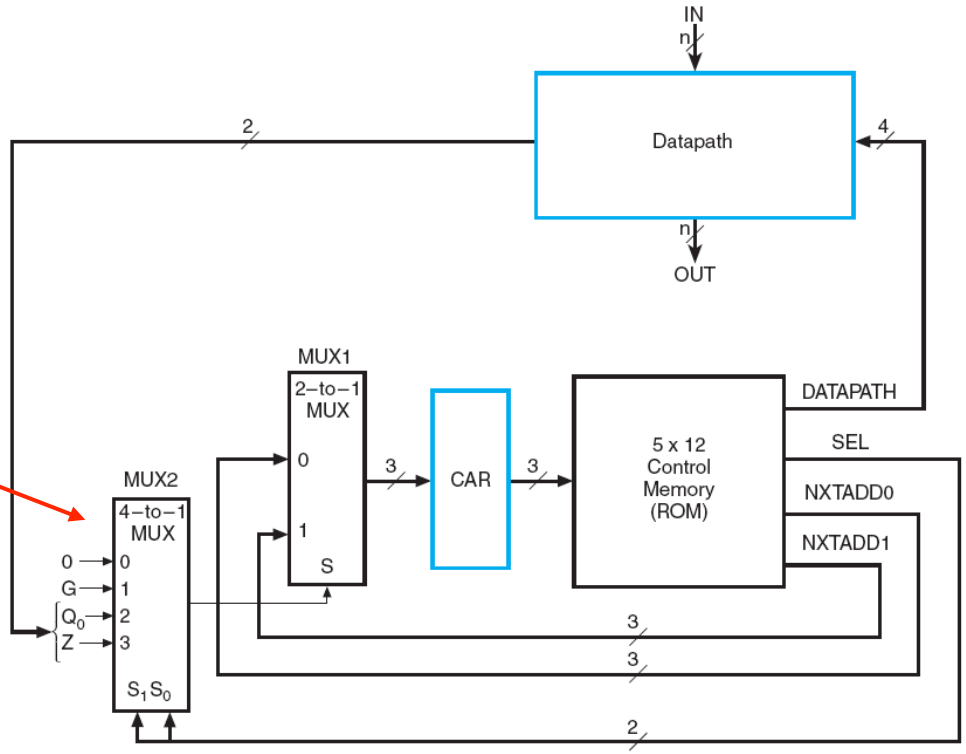
Unidade de Controle Microprogramada para o Multiplicador Binário

- A **Memória de Controle** é uma *ROM* com cinco palavras de 12 *bits* (o Diagrama *ASM* mostra que há cinco estados possíveis)
- Quatro dos *bits* de saída da *ROM* vão para as Entradas de Controle do Caminho de Dados
- Os *bits SEL* controlam um *MUX 4 x 1*, que por sua vez controla outro *MUX 2 x 1*, que decide qual dos dois endereços contidos na microinstrução deve ser carregado no *CAR*



Definição do Campo SEL para o Sequenciamento do Controle do Multiplicador

SEL		
Symbolic notation	**Binary Code**	**Sequencing Microoperations**
NXT	00	$CAR \leftarrow NXTADD0$
DG	01	$\overline{G}$: $CAR \leftarrow NXTADD0$ G: $CAR \leftarrow NXTADD1$
DQ	10	$\overline{Q_0}$: $CAR \leftarrow NXTADD0$ Q_0: $CAR \leftarrow NXTADD1$
DZ	11	$\overline{Z}$: $CAR \leftarrow NXTADD0$ Z: $CAR \leftarrow NXTADD1$



•Que endereço carregar no *CAR*? Há quatro situações possíveis para decidir

•Quando *SEL =00*, o *MUX2* seleciona a entrada *0*, que tem o valor *0*. Um *0* na entrada *S* do *MUX1* seleciona ***NXTADD0*** como o próximo endereço

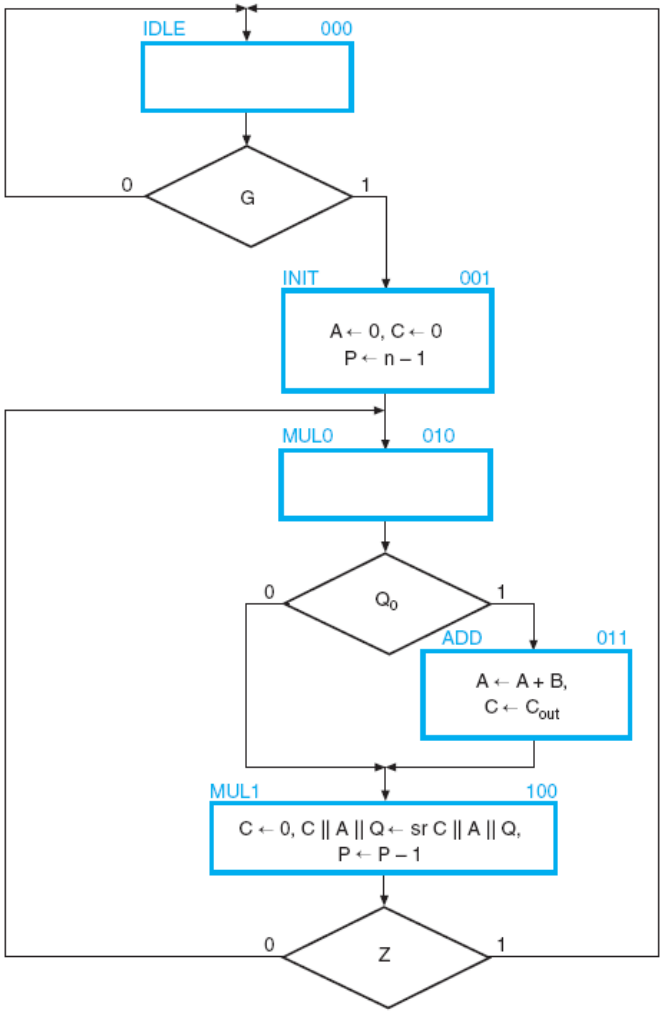
•Quando *SEL=01*, o *MUX2* seleciona a entrada *1*, que é *G*. Se *G=0*, então a entrada *S* em *MUX1* é *0*, selecionando *NXTADD0* como o próximo endereço. Se *G=1*, o próximo endereço deve ser *NXTADD1*

As outras duas variáveis de decisão, Q_0 e Z, operam de forma similar

Microprograma para o Multiplicador Binário em Notação de Transferência de Registrador

Neste microprograma há uma microinstrução para cada estado do Diagrama *ASM*, sendo que o código binário de cada estado corresponde ao conteúdo do *CAR*

Address	Symbolic transfer statement
IDLE	G: $CAR \leftarrow$ INIT, $\overline{G}$: $CAR \leftarrow$ IDLE
INIT	$C \leftarrow 0, A \leftarrow 0, P \leftarrow n-1, CAR \leftarrow$ MUL0
MUL0	Q_0: $CAR \leftarrow$ ADD, $\overline{Q_0}$: $CAR \leftarrow$ MUL1
ADD	$A \leftarrow A + B, C \leftarrow C_{out}, CAR \leftarrow$ MUL1
MUL1	$C \leftarrow 0, C \Vert A \Vert Q \leftarrow$ sr $C \Vert A \Vert Q$, Z: $CAR \leftarrow$ IDLE, $\overline{Z}$: $CAR \leftarrow$ MUL0, $P \leftarrow P-1$



Microprograma Simbólico

Address	NXTADD1	NXTADD0	SEL	DATAPATH	Address	NXTADD1	NXTADD0	SEL	DATAPATH
IDLE	INIT	IDLE	DG	None	000	001	000	01	0000
INIT	—	MUL0	NXT	IT, CC	001	000	010	00	0101
MUL0	ADD	MUL1	DQ	None	010	011	100	10	0000
ADD	—	MUL1	NXT	LD	011	000	100	00	0010
MUL1	IDLE	MUL0	DZ	CC, SD	100	000	010	11	1100